VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Harri Hietamäki

MOOTTORIRAKENTEIDEN VERTAILU SUUNNITELUJÄRJESTELMÄN JA TUOTANNONOHJAUSJÄRJESTELMÄN VÄLILLÄ

Tekniikka ja liikenne
2012

ALKUSANAT

Tämä opinnäytetyö on osa Vaasan ammattikorkeakoulun tietotekniikan koulutusohjelmaa. Työn ajankohta sijoittuu 2012 vuoden syksylle. Työn ohjaajana Vaasan ammattikorkeakoulun puolelta oli yliopettaja Ghodrat Moghadampour. Ohjaajana ABB Oy Motors and Generatorsin puolelta toimi suunnittelupäällikkö Sinikka Sauna-aho.

Haluan kiittää suuresti molempia ohjaajiani sekä järjestelmäkehitysosaston työntekijöitä Vesa Lappalaista, Juha Lainetta ja Antti Helinkoa. Kyseiset henkilöt auttoivat työn etenemisessä ja materiaalin löytämisessä. Erityiskiitokset vielä Jesse Tervolle, jonka ansiosta VBA-koodi saatiin aluille.

Vaasassa 2.12.2012

Harri Hietamäki

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Harri Hietamäki
Opinnäytetyön nimi	Moottorirakenteiden vertailu suunnittelujärjestelmän ja tuotannonohjausjärjestelmän välillä
Vuosi	2012
Kieli	suomi
Sivumäärä	71 + 1 liitettä
Ohjaaja	Ghodrat Moghadampour

Tämä työ on tehty ABB Oy Motors and Generators liiketoimintayksikön järjestelmäkehitysosastolle. Osastolla käytetään tuote- ja suunnittelutiedon hallintajärjestelmänä Teamcenter-ohjelmistoa ja tuotannonohjausjärjestelmänä SAP-ohjelmistoa. Työn tavoite oli suunnitella ja toteuttaa uusi sovellusohjelma, joka löytäisi SAP- ja Teamcenter-järjestelmien välillä eriävät moottorirakenteet.

Rakenteiden haku toteutettiin Teamcenteristä tietokantakyselyillä. SAPissa luotiin uudet rakenteet varianttikonfiguraattorilla niistä moottorikoodeista, jotka Teamcenter tietokantakyselyssä löytyi. Lopputulos saavutettiin vertailemalla luotuja rakenteita alkuperäisin rakenteisiin.

Sovelluksessa hyödynnetään VBA (Visual Basic for Applications) -ohjelmointikielen makrokoodausta luotaessa uudet rakenteet SAPin varianttikonfiguraattorilla. Teamcenterin tietokannasta saatiin moottorirakenteet käyttämällä Java-ohjelmointikielen sisällä SQL (Structured Query Language) -kyselyitä. Sovellus toteutettiinkin pääsääntöisesti Javalla, jolla luotiin myös graafinen käyttöliittymä. Sillä suoritettiin myös itse vertailu ja sen tulostus.

Sovellus saatiin toimimaan ja se vastaa pääsääntöisesti sille asetettuja vaatimuksia. Vertailu paljasti, että moottoreiden osaluettelorakenteet on rakennettu hieman eri tavoin järjestelmissä. Tästä johtuen jokaista eriävää rakenneriviä joudutaan tarkkailemaan yksilökohtaisesti. Lopputulos paikantaa kuitenkin mahdolliset virheet rakenneriveissä. Sovelluksen koko potentiaalia ei ole vielä käytetty ja sen kehittämistä tullaan jatkamaan tulevaisuudessa.

Avainsanat Teamcenter, SAP, moottorirakenteet, vertailu, sovellus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Harri Hietamäki
Title	Comparison of Motor BOMs between ERP and PDM System
Year	2012
Language	Finnish
Pages	71 + 1 Appendices
Name of Supervisor	Ghodrat Moghadampour

This thesis is made for ABB Oy Motors and Generators system development unit. The unit uses Teamcenter software as product data management system and SAP software as enterprise resource planning system. The main goal was to design and implement application software which would discover differences in motor BOMs (bill of materials) between the systems.

BOM search from Teamcenter was implemented with database queries. New BOMs were created with variant configurator in SAP system by using same motor codes as found from Teamcenter database. The result was achieved by comparing the created BOMs to original BOMs.

In application software is utilized VBA (Visual Basic for Applications) programming language for creating new BOMs with SAP variant configurator. BOMs from TC database are found by using integrated SQL (Structured Query Language) queries in Java programming language. The software is mainly programmed with Java like its graphical user interface. Comparison of BOMs and the result writing was also made with Java.

Application is working and it is mostly similar to its requirements. Motor BOMs are not similar between the systems which were discovered in comparison. For that reason every BOM line has to be checked individually. The result still locates potential errors in BOM lines. Whole potential of this application is still not used and the application will be developed in near future.

Keywords Teamcenter, SAP, motor BOMs, comparsion, application

KÄYTETYT LYHENTEET JA MERKIT

ABB	Asea Brown Boveri
ArrayList	Java-ohjelmoinnissa käytettävä listarakenne, joka sallii elementtien lisäämisen ja poistamisen.
BOM	Bill Of Material, osaluettelorakenne
HashMap	Java-ohjelmoinnissa käytetty hajautustaulu, jonka hakurakenne yhdistää avaimen sen arvopariin.
Java	Olioihin perustuva ohjelmointikieli
Luokka	Määrittää yhteiset piirteet tietylle joukolle olioita
Makro	Sovellusohjelmien laji, jossa ohjataan automaattisesti käyttäjän toimintaa
Metodi	Luokan sisällä suoritettava aliohjelma, joka käsittelee olion tietoa
Olio	Ohjelman perusyksikkö, joka sisältää loogisesti yhteenkuuluvaa tietoa ja toiminnallisuutta
Rakennerivi	Tutkittava rivi, joka sisältää moottorin osan tuotekoodin, sekvenssinumeron ja kappalemäärän.
SAP	Systems, Applications and Products in Data Processing, ABB:llä käytössä oleva tuotannonohjausjärjestelmä
SQL	Structured Query Language, kyselykieli
TC, Teamcenter	ABB:llä käytössä oleva tuote- ja suunnittelutiedon hallintajärjestelmä
Transaktio	Kutsuu ja ajaa SAPin ABAP-koodia
Varianttikonfiguraattori	Luo varianttikoodien perusteella moottorirakenteen tiettyjä sääntöjä noudattaen
Varianttikoodi	Moottorikoodin perässä oleva lisäkoodi

VBA	Visual Basic for Applications, ohjelmointikieli, joka on tehty Microsoftin ohjelmistoihin.
ZCU50	Transaktio, joka simuloi moduulivariantteja.

SISÄLLYS

1 JOHDANTO

1.1 Tutkimuksen pohjatietoa

ABB Oy Motors and Generatorsin liiketoimintayksikössä käytetään tilausten moottorirakenteiden konfiguroimiseen SAP-tuotannonohjausjärjestelmässä olevaa varianttikonfiguraattoria. Varianttikonfiguraattori luo moottorirakenteen annettujen varianttikoodien perusteella. SAPiin on rakennettu transaktio ZCU50, jonka avulla SAPin varianttikonfiguraattoria voidaan käyttää myös erillisenä toimenpiteenä. Tämän avulla voidaan saada moottorin rakenne luoduksi, kun syötetään kyseiseen transaktioon moottorikoodi ja tarvittavat varianttikoodit. Transaktion tuloksena on mahdollista saada moottorin moduulilista taulukkoon Excel-muodossa.

Tuote- ja suunnittelutiedon hallintajärjestelmänä yksikössä toimii Teamcenter. Teamcenteriä käyttää pääosin suunnittelijat ja se onkin suunnittelutyön pääpaikka sisältäen liittymät suunnitteluohjelmiin. Sen tietokannasta löytyy moottorikoodeja, joiden rakenteita ei ole luotu konfiguraattorin sääntöjen mukaan. Tästä johtuen moottoreiden rakenteissa saattaa olla virheellistä tai vanhentunutta tietoa. Rakenteet eivät päivity automaattisesti Teamcenterin puolelle ja pysyvät siellä virheellisinä, jos SAPin konfiguraattoriin tulee sääntömuutoksia.

1.2 Tavoitteet

Tämän työn tavoite on saada paikannettua eroavaisuudet moottorirakenteissa Teamcenter-järjestelmän ja SAP-järjestelmän välillä. Teamcenterin tietokannasta pitäisi ensimmäiseksi saada ulos kaikki moottorikoodit, joita voidaan syöttää SAPin konfiguraattorille. ZCU50-transaktiossa toimiva konfiguraattori pitiäsi automatisoida siten, että suurempi määrä Teamcenterin moottorikoodeja vietäisiin sille yhdellä kertaa ja se pystyisi simuloimaan kaikkiin uudet rakenteet. Uudet rakenteet pitäisi vielä saada talteen, että simuloinnista olisi hyötyä.

Simulointituloksen taltioinnin jälkeen tarvitaan tieto siitä, mitkä rakennerivit ovat muuttuneet simuloinnin aikana ja miten moottorin osaluettelorakenne on esitetty molemmissa järjestelmissä. Tämä toteutettaisiin siten, että Teamcenterissä olevia

alkuperäisiä rakenteita vertailtaisiin SAPissa luotuihin rakenteisiin ohjelmointia hyödyntäen. Vertailu tapahtuisi moottorikohtaisesti rakennerivi kerrallaan.

Vertailun jälkeen tarvitaan vielä lopputuloksen esittäminen käyttäjälle selkeään ja ymmärrettävään muotoon. Sen toteutustapana toimisi hyvin Excel-taulukko, johon molempien järjestelemien rakenteet voitaisiin tulostaa rinnakkain ja yhdenmukaiset rakennerivit kohdakkain. Lisäksi eriävät rakennerivit täytyisi saada käyttäjälle esitettyä selkeästi.

1.3 Ongelmat

Ongelmana on järjestelmien erilaisuus, mikä vaatii tarkempaa tutustumista niiden toiminnallisuuteen. Tutustuminen muodostuu tärkeäksi varsinkin Teamcenterin puolella. Tietokantaan pitäisi ensin rakentaa yhteys ja lisäksi selvittää, miten rakenteet ovat siellä ja millaisilla keinoilla ne saa sieltä ulos.

SAPissa on transaktio CS03, joka näyttää kiinteiden moottoreiden rakenteet. Nämä rakenteet eivät kuitenkaan ole ajan tasalla, koska konfiguraattoriin on tehty muutoksia, eikä niitä ole sen jälkeen päivitetty kiinteisiin rakenteisiin. Tästä syystä SAPin rakenteita ei saada tietokannasta ulos vastaavalla tavalla kuin Teamcenterin puolella.

Tähän ratkaisuna on yllämainittu ZCU50-transaktio. Sen suurimmaksi ongelmaksi muodostuukin se, miten saada transaktio toimimaan automaattisesti. Automaattisuus tässä tarkoittaa sitä, että transaktiossa olevat pakolliset kentät täyttyisivät itsestään, eikä käyttäjän tarvitse muuta kuin odottaa simuloinnin valmistumista. Tuloksen talteenotto tapahtuisi myös automaattisesti.

ZCU50-transaktion ongelmana on se, että se voi simuloida moottorirakenteita vain yksi kerrallaan. Se on hyvin hidas tapa ja jo yhdenkin moottorin simulointi vie aikaa. Tästä johtuen massamäärien simulointi ei tule kysymykseen ilman automatisointia. Simuloinnin hitaus johtuu monimutkaisista konfiguraattorin säännöistä, jotka on linkattu yksi toisensa taakse SAPin tietokannassa. Simuloinnin nopeuttamiseen ei voida tehdä juuri mitään, mutta simuloinnin voisi suorittaa esimerkiksi viikonloppuna, kun SAPin käyttö on rajoittuneempaa yksikön verkos-

sa. Molemmista järjestelmistä ulossaadut rakenteet vaativat vielä tavan verrata niitä keskenään. Tulostaminen kohdakkain taulukkoon vaatii myös oman logiikkansa.

Suurimpana ongelmana on kuitenkin koko tapahtumaketjun yhtenäinen toteutus. Haastavuutta lisää eri järjstelmät ja mahdolliset eri ohjelmointikielet. Nämä kaikki pitäisi saada erilaisin keinoin kommunikoimaan keskenään.

1.4 Toimeksiantaja

Tämän työn aiheen sain ABB Oy Motors and Generators liiketoimintayksikön järjestelmäkehitysosastolta, jonka pääasiallinen työkalu on Teamcenter. Osaston vastuualueeseen kuuluu jatkuva Teamcenter-järjestelmän rakentaminen ja ylläpitäminen. Osaston tähtäimenä voitaisiin pitää virheetöntä tietokantaa järjestelmässä. Teamcenterin moottorirakenteiden vertailutarkoitukseen työssä tulee ohjelmoida eräänlainen sovellusohjelma, joka tulisi avuksi pääosin suunnittelulle ja järjestelmäkehitysosastolle. Suunnittelu käsittää kaikki Motorsin yksiköt maailmanlaajuisesti. Sovelluksen hyöty ja käyttötarkoitus tulee olemaan helpotus tuoteylläpidossa.

1.4.1 ABB-yhtymä

ABB syntyi vuonna 1988, kun ruotsalainen Asea ja sveitsiläinen Brown, Boveri & Cie fuusioituivat. Kirjainlyhenne ABB tulee sanoista Asea Brown Boveri. Suomessa sähkövoima- ja automaatioteknologioiden erikoisosaamista on kartutettu jo 120 vuoden ajan. ABB:n menestys johtaa sen teknologisesta voimasta ja perustuu vankkoihin paikallisiin juuriin, joita Suomessa edusti aikanaan Strömberg. /3/

”Nykypäivänä ABB on globaali, johtava sähkövoima- ja automaatioteknologiayhtymä. Sen tuotteet, järjestelmät ja palvelut on tehty parantamaan teollisuus- ja energiayhtiöasiakkaiden kilpailukykyä.” /3/ Työntekijöitä ABB:llä on yli 124 000 ja he ovat jakautuneet noin 100:aan eri maahan. Pääkonttori sijaitsee Sveitsissä ja yrityksen pääjohtajana toimii tällä hetkellä Joe Hogan. Suomessa työntekijöitä on yli 7000 ja suurimmat tehdaskeskittymät Suomessa sijaitsevat Vaasassa ja Helsingissä. /3/

Ydinliiketoiminta on organisoitu ABB-yhtymässä viiteen divisioonaan. Nämä divisioonat ovat Automaatiotuotteet, Prosessiautomaatio, Sähkövoimajärjestelmät, Sähkövoimatuotteet ja Robotit. Automaatiotuotteet divisioona käsittää myös Motors and Generators liiketoimintayksikön, jonne tämä tutkimus tehdään. /3/

1.4.2 ABB Oy Motors and Generators

Moottoreiden ja generaattoreiden tuotanto on saanut alkunsa jo vuonna 1889 ja Vaasassa se alkoi vuonna 1944. Nykypäivänä Motors and Generators tarjoaa suuren valikoiman luotettavia korkean hyötysuhteen omaavia sähkömoottoreita ja generaattoreita kaikkiin sovelluksiin. Kuvassa 1 nähdään perinteisiä valurautamoottoreita, joita valmistetaan kokoluokassa 0,25 kW – 1 MW. Seuraavassa on listattuna muita tuotteita, joita Motors valmistaa: /4/

- räjähdysvaarallisten tilojen valurautaisia Ex-moottoreita
- rullaratamoottoreita
- laivamoottoreita
- tuulivoimageneraattoreita

Kuva 1. Valurautaisia oikosulkumoottoreita. /4/

ABB Oy Motors and Generators on maailmanlaajuinen liiketoimintayksikkö ja sillä on tehtaita Suomen lisäksi muualla Euroopassa, Aasiassa, Afrikassa ja Etelä-Amerikassa. Sen lisäksi sillä on keskusvarastoja Euroopassa ja Aasiassa. Suomessa tehtaat sijaitsevat Helsingissä ja Vaasassa. Vaasassa Motors on vielä jakautunut

3:een eri toimipisteeseen. Siellä on emotehdas eli MM-rakennus, jossa järjestelmäkehitysosasto toimii. Lisäksi se kattaa KK-rakennuksen ja suunnittelijoiden toimistotilat KT-rakennuksesta Hietalahden Strömberg Parkissa. /4/

2 MOOTTORIT TIETOKANNASSA

Ennen ohjelman suunnitteluvaihetta täytyy perehtyä hieman tutkittavaan materiaaliin. Tutkimuksen varsinaisena kohteena on moottoreiden rakennerivit Teamcenterissä ja SAPissa. Niiden selvittämiseen vaaditaan kuitenkin ymmärrystä myös moottorin rakenteesta. Tässä luvussa käsitelläänkin moottorin rakennetta sekä rakennerivien muodostumista. Tämän lisäksi perehdytään siihen, miten moottorit ovat tunnistettavissa tietokannassa.

2.1 Moottorikoodit

Moottorikoodit ovat moottoreiden tuotekoodeja ja jokaisella erilaisella moottorilla on oma tuotekoodinsa. Näiden koodien perusteella moottorit yksilöidään tietokannassa. Aiemmin jokainen tilaus käsiteltiin kokonaan yksilöllisenä rakenteenaan, mutta muistinvaraisesti vanhoja rakenteita hyödyntäen. Moottoreiden tuotekoodit voitaisiin jakaa kahteen eri ryhmään. Vakiomoottorit kuuluvat kiinteisiin moottorikoodeihin ja erikoismoottorit varianttikoodillisiin moottorikoodeihin.

2.1.1 Kiinteä moottorikoodi

Kiinteällä moottorikoodilla tarkoitetaan vakioidun rakenteen sisältävää moottoria, jolla on tietty tuotekoodi. Nämä moottorit löytyvät ABB:n tuotevalikoimasta ja niitä kutsutaan yleisesti nimellä vakiomoottori tai perusmoottori. Nämä moottorit ovat saaneet kiinteän moottorikoodin, koska niiden menekki on suuri. Menekistä johtuen moottoreita voidaan tehdä myös varastoon. Koodi itsessään on 14 merkkiä pitkä. Se muodostuu kirjainten ja numeroiden yhdistelmästä. Tämä yhdistelmä ei kuitenkaan ole satunnainen, generoitu merkkijono, vaan jokaisella merkillä tai merkkiyhdistelmällä on tarkoituksensa.

2.1.2 Varianttikoodillinen moottorikoodi

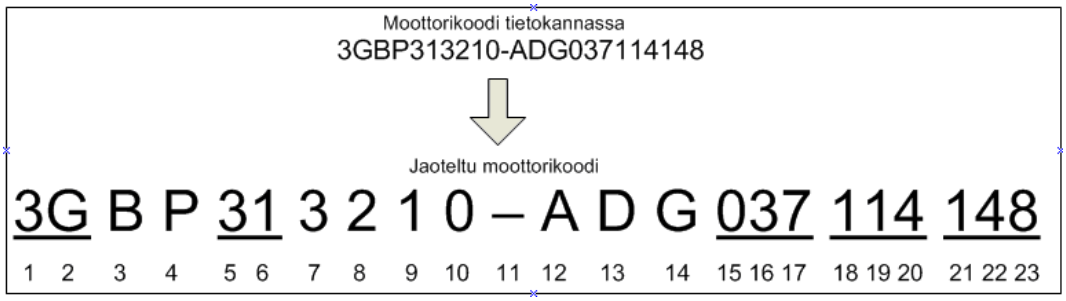
Varianttikoodillinen moottorikoodi muodostuu 14 merkkisestä kiinteästä moottorikoodista ja sen perään liitettävistä varianttikoodeista. Varianttikoodeja käytetään silloin, kun moottoriin liitetään 1 tai useampi lisäosa. Tämä tapahtuu silloin, kun tuotevalikoimasta löytyvää vakiomoottoria joudutaan räätälöimään jonkun tai use-

amman eri komponentin osalta. Yksinkertaisimmillaan se voi olla vain yhden osan erilaisuus alkuperäisestä moottorista. Varianttikoodilliset moottorikoodit ovat hyvin asiakaskohtaisia. Niitä luodaan silloin, kun vakiomoottoreista ei ole löytynyt sopivaa asiakkaalle. Periaatteena on, että mitä räätälöidympi moottori on, sitä enemmän kiinteän moottorikoodin perässä on myös varianttikoodeja.

Yhden varianttikoodin pituus on aina kolme merkkiä. Näiden kolmen merkin perusteella varianttikonfiguraattori osaa yhdistää jonkun tietyn lisäosan tai tehdä muutoksen moottorin kokoonpanoon. Yksi varianttikoodi voisi tarkoittaa esimerkiksi takometriä lisäosana.

2.2 Tuotekoodin muodostuminen

Tuotekoodi muodostuu vähintään 14 merkkisestä koodista, mutta varianttikoodeista riippuen se voi olla pitempi. Kuvasta 2 ja sen selvityksistä voidaan huomata merkkien tarkoitus. Jotkut tyyppitunnisteet vaativat 1, jotkut 2 merkin kuvauksen ja yllämainitut varianttikoodit tunnistettiin 3 merkistä. Kuvassa on jaettu tietokannasta löytyvä moottorin tuotekoodi pienempiin osiin. Kaikkein alimmainen rivi ilmaisee merkin järjestysnumeron.



Kuva 2. Tuotekoodin muodostuminen.

Tuotekoodin tarkempi läpikäynti kuvan perusteella:

- Merkit 1 ja 2 ilmaisevat yhdessä moottorin tyyppiä.
- Merkki 3 kuvaa moottorin rungon valmistustapaa.
- Merkki 4 kertoo prosessimoottorista.
- Merkit 5 ja 6 ilmaisevat yhdessä moottorin runkokoon eli akselikorkeuden.
- Merkki 7 kertoo napaluvun.

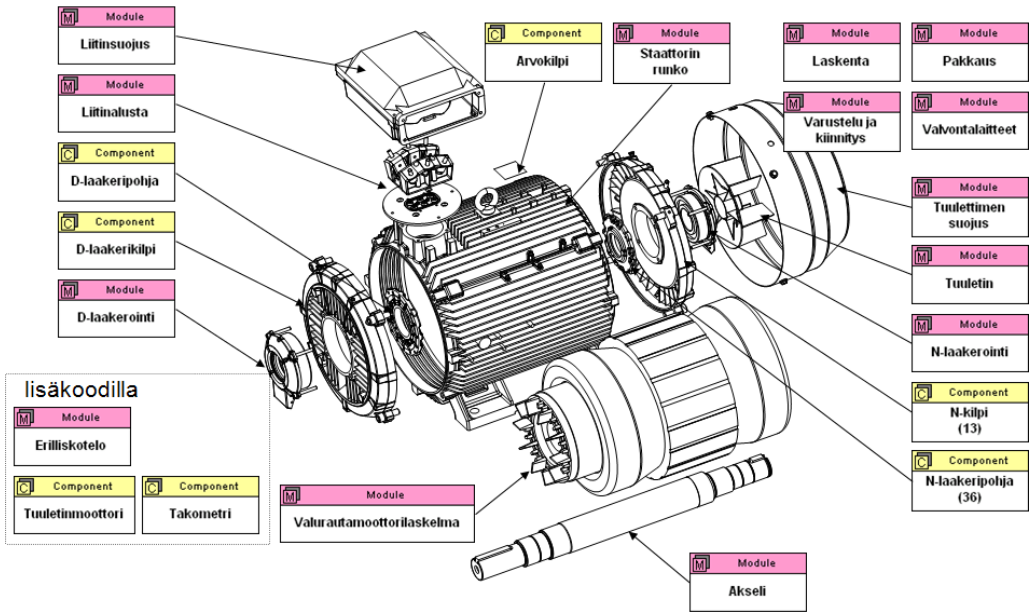
- Merkki 8 ilmaisee rungon pituuden.
- Merkki 9 ilmaisee paketin pituuden.
- Merkki 10 kertoo 1- tai 2-nopeusmoottorista.
- Merkki 11 toimii tuotekoodissa erottimena.
- Merkki 12 tarkoittaa moottorin asennusasentoa.
- Merkki 13 ilmaisee jännitekoodin.
- Merkki 14 ilmaisee sukupolvikoodin.
- Merkit 15, 16 ja 17 ovat ensimmäinen varianttikoodi.
- Merkit 18, 19 ja 20 ovat toinen varianttikoodi.
- Merkit 21, 22 ja 23 ovat kolmas varianttikoodi.

2.3 Moottorin rakenne

Sähkömoottori on monen osan kokonaisuus. Nämä osat muodostavat moottorin rakenteen. Tietokantaan on rakennettu jokaiselle moottorille omanlainen osaluettelorakenne eli BOM. BOM toimii puurakennemaisesti siten, että sillä on päätasolla pääosat ja tietyillä osilla voi olla myös oma alikokoonpano. Alikokoonpanon omaavia osia kutsutaan moduuleiksi. Moottorin moduulit pitää sisällään myös oman osaluettelorakenteen. Tietokannassa moduuli tunnistetaan 3GZF1010-alkuisesta nimikkeistä.

Moduulilla olevaa BOMia kutsutaan super BOMiksi. Tätä voidaan edelleen kutsua toiseksi tasoksi. Toisenkin tason osilla voi olla omia alikokoonpanoja, jolloin puhutaan jo kolmannesta tasosta. Tässä työssä on tarkoitus verrata pelkästään ensimmäisen tason rakennerivejä, mikä tarkoittaa niitä osia, jotka löytyvät suoraan moottorin alta.

Kuvassa 3 voidaan nähdä käytännössä moottorin rakenne. Siinä on esillä vain pääosat eli ensimmäinen taso. Vaaleanpunaisissa ruuduissa on moottorin moduulit, joilla on oma alikokoonpanonsa. Keltaisella kuvaan on merkitty komponentit, jotka ovat yksittäisiä osia ilman alikokoonpanoa. Lisäkoodiruutu tarkoittaa sitä, että varianttikoodeja lisäämällä voidaan saada kyseiset lisäkomponentit tai moduuli.



Kuva 3. Valurautamoottorin moduulit 280-400 kokoluokassa. /5/

2.4 Rakennerivit

Yhdellä moottorin komponentilla tai moduulilla on ominaistietoja. Näitä tietoja on esimerkiksi selkokielinen nimi, nimike tietokannassa, osanumero, paino ja kappalemäärä. Yhtä edellisien tietojen yhdistelmää voitaisiin kutsua kyseisen komponentin tai moduulin rakenneriviksi. Rakennerivit ovat juuri niitä, mitä tässä työssä tullaan vertailemaan Teamcenterin ja SAPin välillä. Rakennerivistä käytetään nimitystä BOM line. Kuvassa 4 nähdään yhden moottorin ensimmäisen tason rakennerivit, jotka koostuvat nimikkeestä tietokannassa, selkokielisestä nimestä, osanumerosta, alikokoonpanotunnuksesta, kappalemäärästä ja yksiköstä.

LKH80813S	CALCULATION	201	N	1	PC
3GZF101017S1	MONITORING MODULE	400	N	1	PC
3GZF123031-32	ROTOR CORE	211	L	1	PC
3GZF183031-1	SHAFT	16	L	1	PC
3GZF101017M1	MONITORING MODULE	4	N	1	PC
3GZF101011M14	SHAFT MODULE	16	N	1	PC
3GZF101012-35	D-BEARING MODULE	18	N	1	PC
3GZF213731-1	END SHIELD	12	L	1	PC
3GZF101024-58	D-INNER BEARING COVER MODULE	34	N	1	PC
3GZF101016-9	TERMINAL BLOCK MODULE	302	N	1	PC
3GZF101015-405	TERMINAL BOX MODULE	301	N	1	PC
3GZF101013-24	N-BEARING MODULE	24	N	1	PC
3GZF203731-69	END SHIELD	13	L	1	PC
3GZF101025-2	N-INNER BEARING COVER MODULE	36	N	1	PC
3GZF101010-9	STATOR FRAME MODULE	11	N	1	PC
3GZF304128-3	FAN	61	L	1	PC
3GZF101026-6	FAN COVER MODULE	67	N	1	PC
3GZF101018-7	OUTFIT MODULE	17	N	1	PC
3GZF101053-2	CONNECTION INTERFACE MODULE	390	N	1	PC
3GZF101020-26	PACKING MODULE	750	N	1	PC

Kuva 4. Rakennerivejä.

3 SUUNNITTELUN JA TUOTETIEDON HALLINTAJÄRJESTELMÄ

ABB Oy Motors and Generators on valinnut käyttöönsä Teamcenter-ohjelmiston PDM-järjestelmäksi eli suunnittelu- ja tuotetiedon hallintajärjestelmäksi. Teamcenter on Siemens PLM Softwaren kehittämä ohjelmisto. Teamcenterissä hoidetaan nimikkeisiin ja tuoterakenteeseen sekä niiden tekniseen kuvaukseen kiinteästi liittyvät toiminnallisuudet. Näitä ovat teknisten nimikkeiden hallinta ja revisiohallinta. Teamcenterissä hoidetaan myös rakenteiden ja osaluetteloiden hallinta, mutta ei siinä tapauksessa, jos ne ovat kertakäyttöisiä työnumeroita.

Lisäksi Teamcenterin toiminnallisuuteen kuuluu piirustusten, dokumenttien ja CAD-datan hallinta. Teamcenteristä löytyykin liittymät AutoCAD, NX ja I-deas tietokoneavusteisiin suunnittelujärjestelmiin. Teamcenter on pääpaikka kaikelle suunnittelun tarvitsemalle tiedolle. Siitä löytyy myös liittymät sähkösuunnittelujärjestelmä Adeptiin ja tuotannonohjausjärjestelmä SAPiin. /9/

3.1 Revisiohallinta

Teamcenterissä on käytössä automaattinen revisiohallinta. Revisio tarkoittaa ajallisesti peräkkäistä ja erilaista versiota. Vain viimeisin hyväksytty revisio on voimassa, eikä kahta samanaikaista revisiota sallita. Tässä työssä vertailtavat moottorirakenteet tulevat aina moottorin uusimmasta revisiosta. Revisiohallinnan toiminta pakottaa käyttäjää tekemään pienimmätkin muutokset aina uuteen versioon. Keskeneräistä versiota, joka ei ole käynyt hyväksymisprosessia läpi, voidaan muokata edelleen. /9/

Revisiohallinta pitää huolen, ettei vanhoihin hyväksyttyihin revisioihin voi enää tehdä korjauksia. Uuden muutoksen tekeminen toimii check in – check out -menetelmällä. Kun suunnittelija aloittaa muutoksen revisiossa, tapahtuu check in -toiminto revisiohallinnassa. Kenelläkään muulla ei tällöin ole muutosoikeuksia, kunnes suunnittelija luovuttaa muutosoikeudet (check out). /9/

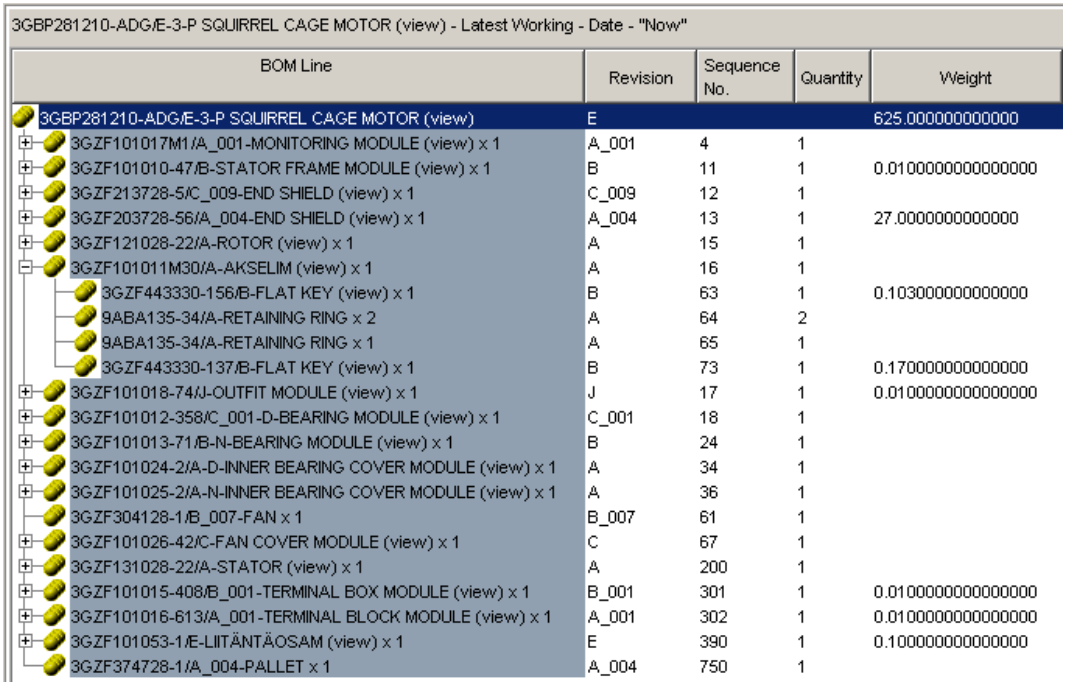
Teamcenterissä on käytössä workflow-ominaisuus, joka hallitsee prosesseja ja työnkulkua sähköisesti. ABB:lla sitä käytetään suunnittelutiedon hyväksymiseen ja automaattiseen julkaisuun SAP-tuotannonohjausjärjestelmässä. Revision hyväksyminen toimii rajapintana Teamcenterin ja SAPin välillä. Revision täytyy käydä hyväksymisprosessi läpi hyväksyjällä ja joskus vielä tarkastajallakin. /9/

3.2 Osaluetteloiden hallinta

Teamcenterissä on osaluetteloiden hallintaan rakennettu osaluetteloeditori PSE eli Product Structure Editor. Se on suunnittelijoiden työkalu, jolla voidaan muokata tai tehdä uusia osaluetteloita. Jokaisella eri revisiolla voi olla erilainen osaluettelo. Osaluetteloista käytetään Teamcenterissä nimitystä BOM. Jokaisella nimikkeellä tietokannassa on osaluettelo, mikäli se sisältää omia komponentteja. Kaikki osaluettelot luodaan ja ylläpidetään PSE:ssä. /9/

Osaluettelo kertoo, millainen esimerkiksi moottorin tuoterakenne on. Siinä on eriteltynä moottorin alikokoonpano, joka kertoo millaisista osista ja komponenteista moottori koostuu. Se sisältää myös tiedon siitä, mitä piirustuksia tarvitaan sen valmistamiseksi ja kuvaamiseksi. Moottorirakenteet on suunniteltu PSE:ssä.

Kuvasta 5 nähdään, että moottori on avattu Teamcenterin PSE:ssä ja kyseisen moottorin osaluettelorakenne on käsiteltävissä. Kuvassa on valittuna moottori, mutta toisella rivillä siinä nähdään ensimmäinen moduuli, sen nimike ja selkokielinen nimi. Lisäksi otsikoista Sequence No ja Quantity voitaisiin poimia moduulille osanumero ja kappalemäärä. Tästä saadaan yksi vertailtava rakennerivi. Kuviosta voidaan huomata myös, että siinä on avattuna akselimoduuli, jonka alta paljastuu 4 akselimoduulille kuuluvaa osaa. Näille toisen tason osille on myös omat rakennerivinsä PSE:ssä.



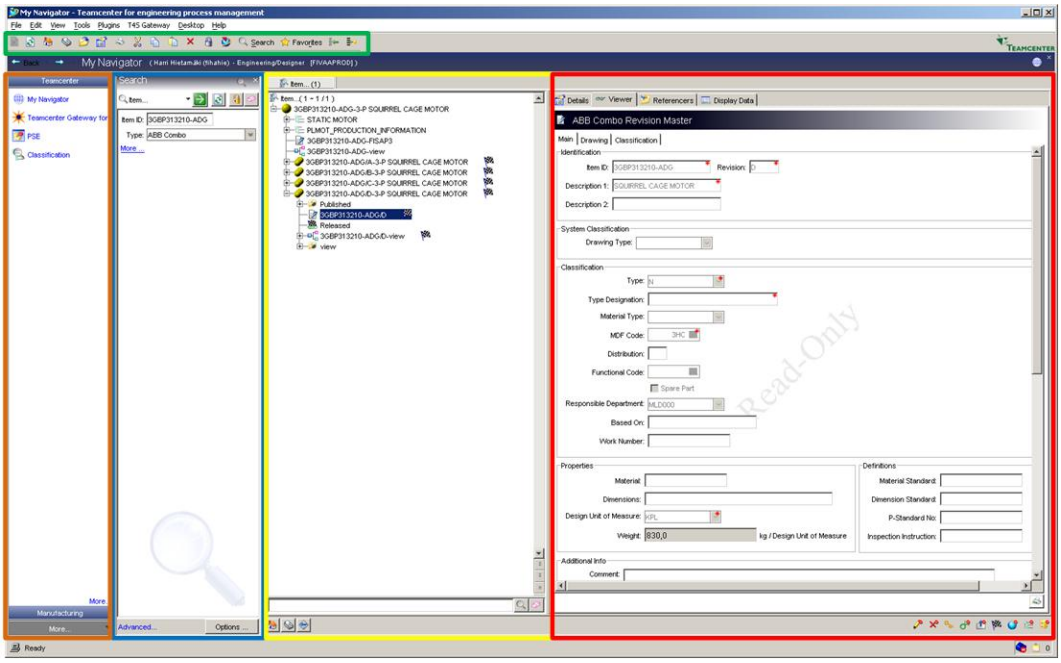
3GBP281210-ADG/E-3-P SQUIRREL CAGE MOTOR (view) - Latest Working - Date - "Now"

BOM Line	Revision	Sequence No.	Quantity	Weight
3GBP281210-ADG/E-3-P SQUIRREL CAGE MOTOR (view)	E			625.000000000000
3GZF101017M1/A_001-MONITORING MODULE (view) x 1	A_001	4	1	
3GZF101010-47/B-STATOR FRAME MODULE (view) x 1	B	11	1	0.0100000000000000
3GZF213728-5/C_009-END SHIELD (view) x 1	C_009	12	1	
3GZF203728-56/A_004-END SHIELD (view) x 1	A_004	13	1	27.0000000000000
3GZF121028-22/A-ROTOR (view) x 1	A	15	1	
3GZF101011M30/A-AKSELIM (view) x 1	A	16	1	
3GZF443330-156/B-FLAT KEY (view) x 1	B	63	1	0.103000000000000
9ABA135-34/A-RETAINING RING x 2	A	64	2	
9ABA135-34/A-RETAINING RING x 1	A	65	1	
3GZF443330-137/B-FLAT KEY (view) x 1	B	73	1	0.170000000000000
3GZF101018-74/J-OUTFIT MODULE (view) x 1	J	17	1	0.0100000000000000
3GZF101012-358/C_001-D-BEARING MODULE (view) x 1	C_001	18	1	
3GZF101013-71/B-N-BEARING MODULE (view) x 1	B	24	1	
3GZF101024-2/A-D-INNER BEARING COVER MODULE (view) x 1	A	34	1	
3GZF101025-2/A-N-INNER BEARING COVER MODULE (view) x 1	A	36	1	
3GZF304128-1/B_007-FAN x 1	B_007	61	1	
3GZF101026-42/C-FAN COVER MODULE (view) x 1	C	67	1	
3GZF131028-22/A-STATOR (view) x 1	A	200	1	
3GZF101015-408/B_001-TERMINAL BOX MODULE (view) x 1	B_001	301	1	0.0100000000000000
3GZF101016-613/A_001-TERMINAL BLOCK MODULE (view) x 1	A_001	302	1	0.0100000000000000
3GZF101053-1/E-LIITÄNTÄOSAM (view) x 1	E	390	1	0.100000000000000
3GZF374728-1/A_004-PALLET x 1	A_004	750	1	

Kuva 5. Oikosulkumoottorin BOM Teamcenterin PSE:ssä.

3.3 Käyttöliittymä

Käyttöliittymä toimii yhdessä ikkunassa, mutta sen taustalle avautuu myös komentoikkuna, joka luo tietokantayhteyden Teamcenteriin. Käyttöliittymä avautuu paremmin selvittämällä sen yleisimmät ominaisuudet käyttöliittymäkuvan kuvan 6 mukaan. Kuvioon on ylhäälle vihreällä merkitty työkalupalkki, joka sisältää myös liittymät NX ja I-deas piirustusohjelmiin. Vasemmalla oranssissa ruudussa on sovelluspalkki, joka näyttää käytössä olevat sovellukset. Yksi käytössä olevista sovelluksista on My Navigator, johon Teamcenterin käyttäminen pääsääntöisesti pohjautuu. Sininen ruutu pitää sisällään Search-työkalun, jolla on kuviossa etsitty moottoria. Keltainen ruutu on osa Search-toimintoa. Siinä on avautunut hakemistopalkki, jolla voi etsiä, käsitellä ja hallita eri objekteja Teamcenterin tietokannassa. Punainen ruutu sisältää katseluikkunan. Katseluikkuna avautuu valitsemalla jokin objekti hakemistopalkista. Kuviossa on moottorin lomake auki, jossa on tarkempia tietoja haetusta moottorista eri välilehdissä. Siinä voidaan katsella vanhoja tai keskeneräisiä revisioita. Katseluikkunaan aukeaa myös esimerkiksi pdf-tiedostot ja pyöriteltävät 3D-piirustukset. /9/



Kuva 6. Teamcenterin käyttöliittymä.

3.4 Virheet tietokannassa

Teamcenterissä olevat kokonaiset moottorirakenteet ovat suurimmaksi osaksi tulleet latauksen yhteydessä, missä vanhan EMIS-tuotannonohjausjärjestelmän data siirrettiin Teamcenteriin. Uudessa tuotannonohjausjärjestelmässä SAPissa on vuoden 2009 jälkeen tullut konfiguraattorisääntöihin muutoksia. Rakenteiden valikoituminen voi poiketa tämänhetkisestä rakenteesta. Uudella tavalla valikoituvia rakenteita ei ole päivitetty Teamcenterin tietokantaan, joten se ei ole ajan tasalla.

Siirron jälkeen on jonkin verran luotu kokonaisia moottorirakenteita perusmoottoreille. Perusmoottoreita luodaan silloin, kun niitä varastoidaan. Tämä joudutaan tekemään sen takia, että SAPissa varastoitavilla moottoreilla täytyy olla nimike. Nimikkeet perustetaan aina Teamcenterissä. Asiakaskohtaisille moottoreille on luotu myös nimikkeitä Teamcenterissä, mutta melko rajoitetusti.

4 TUOTANNONOHJAUSJÄRJESTELMÄ

Koko Suomen ABB on valinnut yhteiseksi tuotannonohjausjärjestelmäksi SAP-ohjelmiston. Sitä voidaan käyttää kaikkiin liiketoimintaprosesseihin liittyvissä tiedonhallinnoissa. /10/ Motorsilla SAPin käyttötoimintoihin kuuluu materiaalivirtojen hoito sekä tuotannonohjaus. Näitä ovat tuotteisiin liittyvien nimikkeiden ja materiaalien hankinnat eli hankinta-aloitteet. Työnumeron osien ja osaluetteloiden sarjanumerointi kuuluvat SAPin tehtäviin. Sen tärkeimpiä tehtäviä on vaiheistus, jonka tehtävänä on vastata kysymyksiin mitä tehdään, missä tehdään ja missä järjestyksessä. Myös vaiheiden kesto, kokonaiskesto ja niiden ajoitus liittyvät tuotannonohjausjärjestelmän toimiin. /9/ Tässä ovat SAPin keskeisimmät hyödyt ABB.llä:

- SAP tukee kaikkia liiketoiminnan prosesseja.
- Eri prosessien tietovirrat yhdistetään yhteiseksi tietokannaksi.
- Toimitusajat nopeutuvat.
- Tiedon analysointi helpottuu, jolloin asiakkaiden tarpeen ennakointi tarkentuu.
- Parhaat toimintatavat tunnistetaan ja ne saadaan yrityksen käyttöön.
- Tieto osaajista keskittyy, jolloin he ovat siellä missä heitä tarvitaan.
- Työssä tarvittavat tiedot löytyvät yhdestä paikasta.
- Työtavat kehittyvät ja yhtenäistyvät. /10/

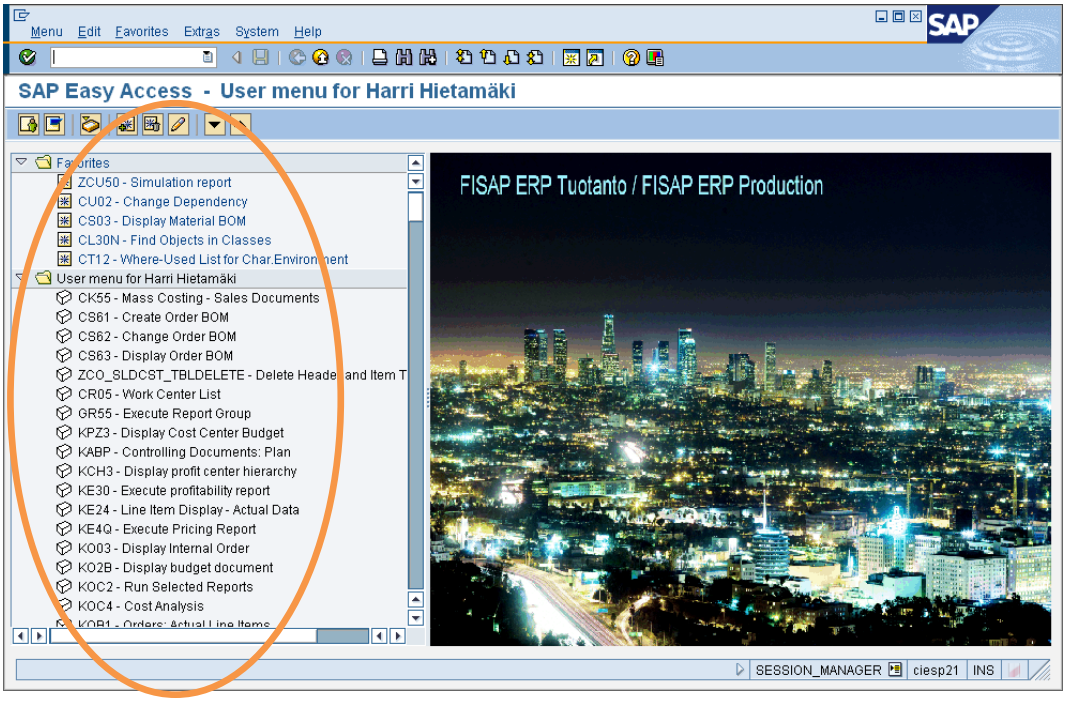
SAP-järjestelmässä voi käyttää eri rakenteita edustamaan yrityksen juridista ja organisatorista rakennetta. Rakenne voidaan määritellä talouden, materiaalihallinnan tai myynnin ja jakelun näkökulmasta. Näitä rakenteita voidaan myös yhdistellä. Organisatoriset rakenteet muodostavat kehykset kaikille liiketoiminnan tapahtumille. /10/

4.1 Käyttöliittymä ja transaktiot

SAPin käyttö perustuu transaktioihin. Transaktio tarkoittaa tietynlaista tietokantatapahtumaa. Tällainen tapahtuma on osa tietokannassa peräkkäin suoritettavia tapahtumia ja hakuja. Jokaisella transaktiolla on oma tarkoituksensa SAP-

ympäristössä. Transaktiot voivat olla SAPin omia tai käyttäjän määrittelemiä. Yhden transaktion tehtävänä voi olla esimerkiksi kaupan luominen.

Kuvassa 7 nähdään SAPin aloitusruutu ja sen vasemmassa reunassa on lista transaktioista. Transaktion tunnisteet muodostuvat muutamasta kirjaimen ja numeron yhdistelmästä. Tämän yhdistelmän vieressä on lisäksi kuvaus, joka kertoo transaktion käyttötarkoituksen. Ne voidaan käynnistää joko listasta painamalla tai syöttämällä transaktion tunnus yllä olevaan tekstikenttään.



Kuva 7. SAPin pääikkuna ja transaktiolista.

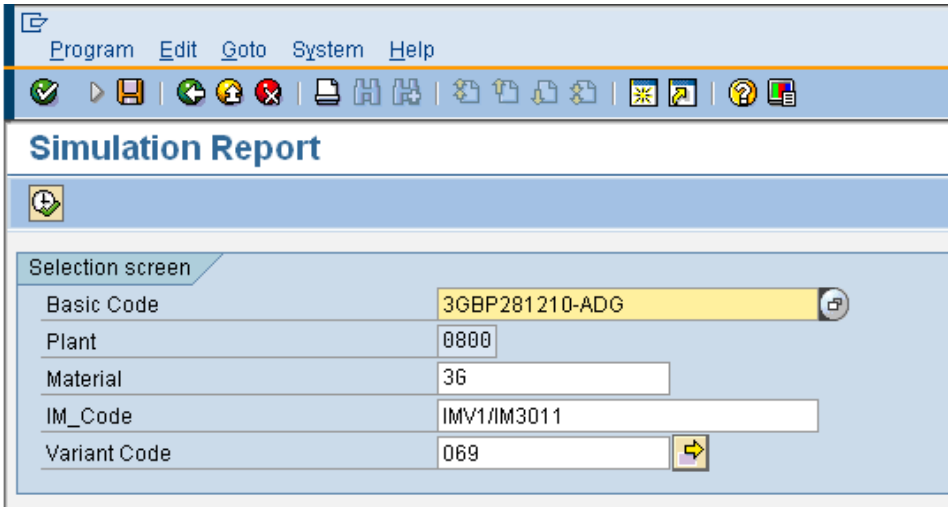
4.2 ZCU50 -transaktio

ZCU50-transaktio luotiin siihen käyttötarkoitukseen, että käyttäjä voi saada listan moduulivarianteista. Moduulivariantti tarkoittaa määriteltyä osakokonaisuutta, joka on kohdistettu tietylle moottorityypille tietyllä rakenteella. Tällä transaktiolla voidaan simuloida vain 3G- ja 3GA-moottoreita. Se on räätälöity SAP Finlandia varten. /10/ Simuloinnin tuloksen saa ulos SAPista ja saatu data tallentuu Excel-ohjelmiston formaattiin taulukoksi.

Tässä on otettava huomioon, että simulointi tapahtuu vain yksi moottorikoodi kerrallaan. Jokaisessa transaktion ruudussa, koodin täytyy tehdä oikeat toiminnot päästäkseen etenemään. Siksi transaktioon on tärkeä tutustua erittäin tarkasti.

4.2.1 Simuloinnin aloitus

Kuvassa 8 nähdään ZCU50-transaktion aloitusruutu. Se pyytää pohjatiedoksi 5 täytettävää kohtaa, jotka ovat Basic Code, Plant, Material, IM Code ja Variant Code. Kaikki kentät, Variant Code -kenttää lukuun ottamatta, on pakko täyttää simulointia varten.



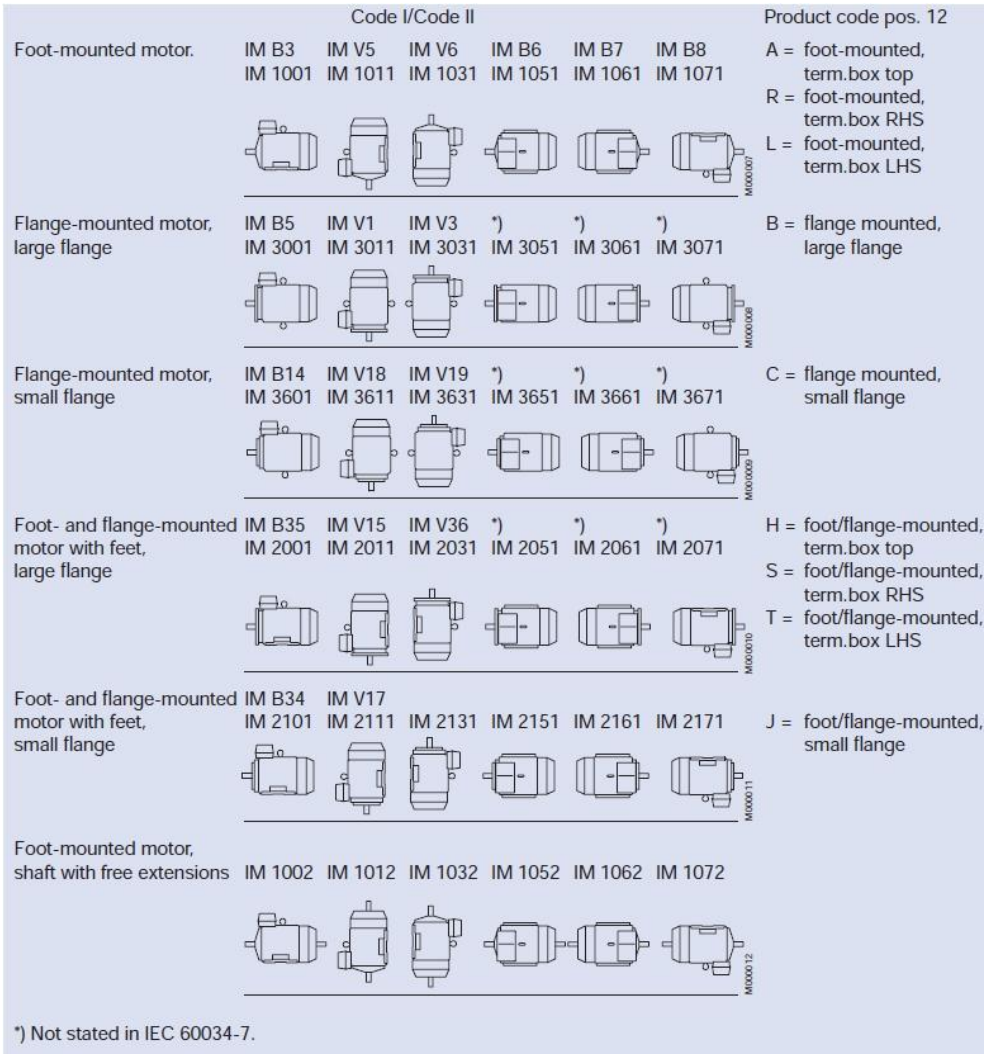
Kuva 8. ZCU50-transaktion aloitusruutu.

Ylimpänä aloitusruudussa on kohta Basic Code, joka tarkoittaa kiinteän moottorin peruskoodia. Tähän tekstikenttään syötetään käsin haluttu peruskoodi tai vaihtoehtoisesti avataan lista. Jälkimmäinen keino listaa kaikki kiinteiden moottoreiden peruskoodit, joita sillä voidaan simuloida. Moottorit tulevat erillisestä Z-taulusta ja niitä on reilu 300 kappaletta.

Toisena kohtana on Plant. Tähän tekstikenttään syötetään tässä työssä aina arvo 0800. Tämä tarkoittaa toimipistettä, joka on tyypillisesti varasto tai tehdas. Muita koodeja olisi itsenäinen juridinen yhtiö 0010, tuotteiden myynti ja jakelu 0080 tai oston organisaatio samalla numerolla 0080.

Material-kohta tarkoittaa moottorin rungon valmistusmateriaalia. Siihen voidaan syöttää koodi 3G, joka tarkoittaa valurautamoottoria. Toinen mahdollisuus on syöttää 3GA, joka tarkoittaa teräslevyrunkoista moottoria. Tämä tutkimus käsittelee vain 3G-moottoreita.

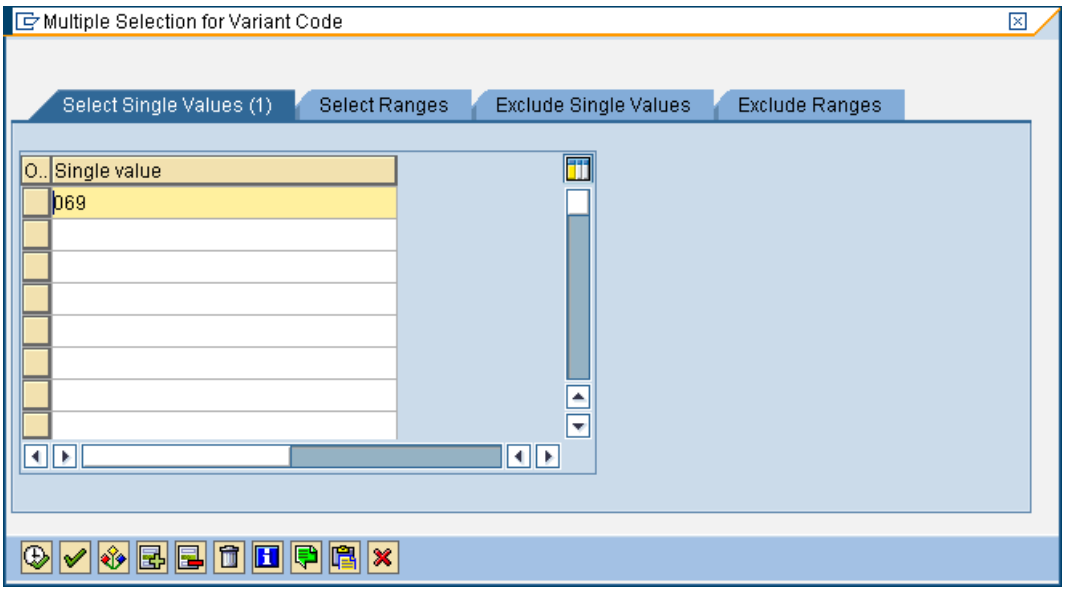
IM Code -kohtaan pyydetään moottorin asennusasentoa. Moottorin asennusasento vaikuttaa simulointitulokseen, koska tietyt moottorin osat voivat vaihtua asennustapaa muuttamalla. Transaktiossa saa listan kaikista mahdollisista asennustavoista, mutta kuvassa 9 on selvitetty, mitä mikäkin koodi käytännössä tarkoittaa.



Kuva 9. IM-koodit. /2/

Variant Code tarkoittaa varianttikoodia. Tämä koodi erottaa samanlaiset moottorit, joita on yhtä tai useampaa alkuperäisen moottorin osaa muuttamalla konfigu-

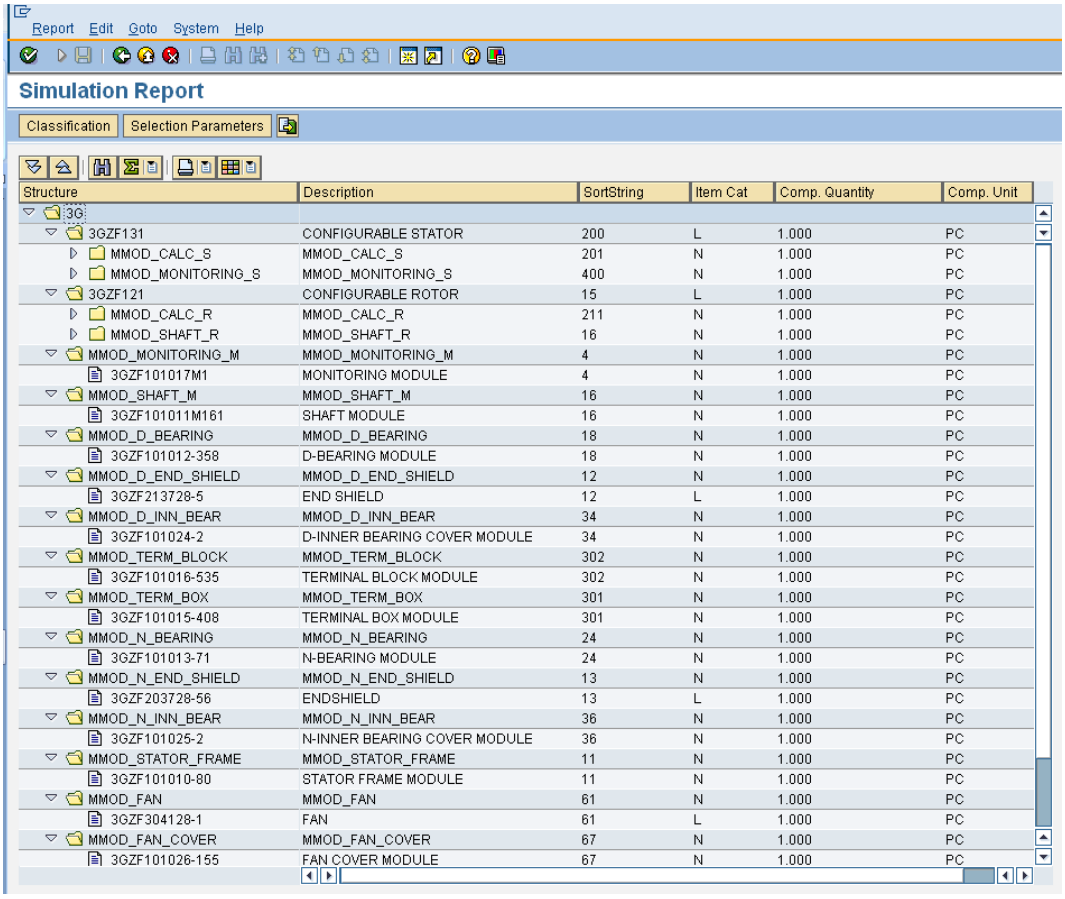
roitu erilaisiksi. Kenttään voidaan syöttää suoraan arvo, mutta useamman variant-tikoodin syöttämiseen vaaditaan erillinen syöttöruutu (Kuva 10.). Se on oma ik-kunan, joka kantaa nimeä Multiple Selection for Variant Code. Kyseisen ikkunan listaan voidaan sitten lisätä useampia varianttikoodeja.



Kuva 10. Varianttikoodien syöttäminen.

4.2.2 Simuloinnin tulos

Kun kaikki läpikäydyt kentät on täytetty, voidaan simulointi suorittaa. Simuloin-nin kesto on yhdestä kahteen minuuttiin. Kuvassa 11 nähdään esimerkki simu-loinnin tuloksesta, joka on saatu aloitusruudussa täytettyjen tietojen perusteella (Kuva 8.). Tuloksessa on moottorin rakennerivejä. Kansiot joudutaan avaamaan käsin, että nähdään osan nimike tietokannassa.



Structure	Description	SortString	Item Cat	Comp. Quantity	Comp. Unit
3G					
3GZF131	CONFIGURABLE STATOR	200	L	1.000	PC
MMOD_CALC_S	MMOD_CALC_S	201	N	1.000	PC
MMOD_MONITORING_S	MMOD_MONITORING_S	400	N	1.000	PC
3GZF121	CONFIGURABLE ROTOR	15	L	1.000	PC
MMOD_CALC_R	MMOD_CALC_R	211	N	1.000	PC
MMOD_SHAFT_R	MMOD_SHAFT_R	16	N	1.000	PC
MMOD_MONITORING_M	MMOD_MONITORING_M	4	N	1.000	PC
3GZF101017M1	MONITORING MODULE	4	N	1.000	PC
MMOD_SHAFT_M	MMOD_SHAFT_M	16	N	1.000	PC
3GZF101011M161	SHAFT MODULE	16	N	1.000	PC
MMOD_D_BEARING	MMOD_D_BEARING	18	N	1.000	PC
3GZF101012-358	D-BEARING MODULE	18	N	1.000	PC
MMOD_D_END_SHIELD	MMOD_D_END_SHIELD	12	N	1.000	PC
3GZF213728-5	END SHIELD	12	L	1.000	PC
MMOD_D_INN_BEAR	MMOD_D_INN_BEAR	34	N	1.000	PC
3GZF101024-2	D-INNER BEARING COVER MODULE	34	N	1.000	PC
MMOD_TERM_BLOCK	MMOD_TERM_BLOCK	302	N	1.000	PC
3GZF101016-535	TERMINAL BLOCK MODULE	302	N	1.000	PC
MMOD_TERM_BOX	MMOD_TERM_BOX	301	N	1.000	PC
3GZF101015-408	TERMINAL BOX MODULE	301	N	1.000	PC
MMOD_N_BEARING	MMOD_N_BEARING	24	N	1.000	PC
3GZF101013-71	N-BEARING MODULE	24	N	1.000	PC
MMOD_N_END_SHIELD	MMOD_N_END_SHIELD	13	N	1.000	PC
3GZF203728-56	ENDSHIELD	13	L	1.000	PC
MMOD_N_INN_BEAR	MMOD_N_INN_BEAR	36	N	1.000	PC
3GZF101025-2	N-INNER BEARING COVER MODULE	36	N	1.000	PC
MMOD_STATOR_FRAME	MMOD_STATOR_FRAME	11	N	1.000	PC
3GZF101010-80	STATOR FRAME MODULE	11	N	1.000	PC
MMOD_FAN	MMOD_FAN	61	N	1.000	PC
3GZF304128-1	FAN	61	L	1.000	PC
MMOD_FAN_COVER	MMOD_FAN_COVER	67	N	1.000	PC
3GZF101026-155	FAN COVER MODULE	67	N	1.000	PC

Kuva 11. 3GBP281210-ADG069 moottorin ensimmäinen taso.

4.2.3 Tuloksen hyödyntäminen

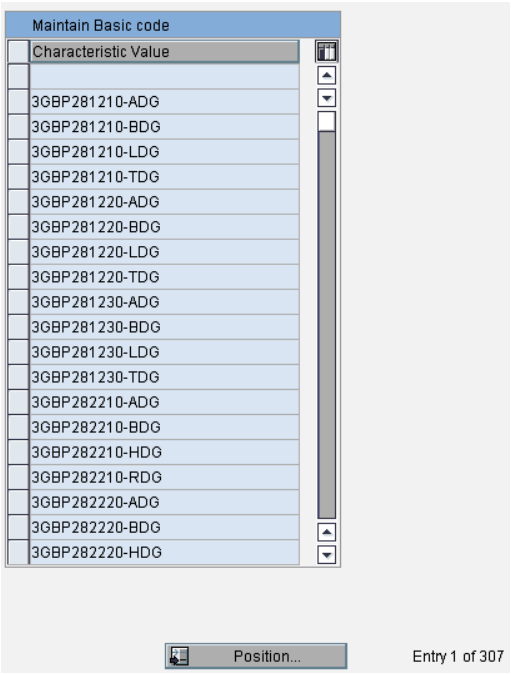
Simuloinnin jälkeen huomataan, että kuvan 11 yläosaan on ilmestynyt 3 painiketta. Niillä on mahdollisuus nähdä klassifiointitiedot, parametrien tiedot tai tallentaa moduulivarianttilista Exceliin. Kuvassa 12 nähdään, kuinka moottorin rakenteet tallentuvat Excelin soluihin. Tähän taulukkoon on kuitenkin tallentunut paljon ylimääräistä, mitä ei tässä työssä tarvita. Tässä kohtaa VBA-koodin hyöty tulee myös esiin, koska sen voi ohjelmoida rakentamaan Excel-taulukko halutulla tavalla. Haluttuun tapaan tulostuisi vain moottorin tuotenumero ja sen ensimmäisen tason tuotekoodit, sekvenssinumero ja kappalemäärä.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
2	Component	SortString	Item Cat	Quantity	Unit	Component	SortString	Item Cat	Quantity	Unit	Component	SortString	Item Cat	Quantity	Unit	Node Level
3	3GZF131	200	L	1.000	PC	MMOD_CALC_S	201	N	1.000	PC	LKH814525	201	N	1.000	PC	3
4	3GZF131	200	L	1.000	PC	MMOD_MONITORIN	400	N	1.000	PC	3GZF10101751	400	N	1.000	PC	3
5	3GZF121	15	L	1.000	PC	MMOD_CALC_R	211	N	1.000	PC	3GZF123028-5	211	L	1.000	PC	3
6	3GZF121	15	L	1.000	PC	MMOD_SHAFT_R	16	N	1.000	PC	3GZF183028-13	16	L	1.000	PC	3
7						MMOD_MONITORIN	4	N	1.000	PC	3GZF101017M1	4	N	1.000	PC	2
8						MMOD_SHAFT_M	16	N	1.000	PC	3GZF101011M161	16	N	1.000	PC	2
9						MMOD_D_BEARING	18	N	1.000	PC	3GZF101012-358	18	N	1.000	PC	2
10						MMOD_D_END_SHIE	12	N	1.000	PC	3GZF223028-8	12	L	1.000	PC	2
11						MMOD_D_INN_BEAR	34	N	1.000	PC	3GZF101024-2	34	N	1.000	PC	2
12						MMOD_TERM_BLOCK	302	N	1.000	PC	3GZF101016-535	302	N	1.000	PC	2
13						MMOD_TERM_BOX	301	N	1.000	PC	3GZF101015-408	301	N	1.000	PC	2
14						MMOD_N_BEARING	24	N	1.000	PC	3GZF101013-71	24	N	1.000	PC	2
15						MMOD_N_END_SHIE	13	N	1.000	PC	3GZF203728-56	13	L	1.000	PC	2
16						MMOD_N_INN_BEAR	36	N	1.000	PC	3GZF101025-2	36	N	1.000	PC	2
17						MMOD_STATOR_FRA	11	N	1.000	PC	3GZF101010-48	11	N	1.000	PC	2
18						MMOD_FAN	61	N	1.000	PC	3GZF304128-1	61	L	1.000	PC	2
19						MMOD_FAN_COVER	67	N	1.000	PC	3GZF101026-155	67	N	1.000	PC	2
20						MMOD_EQUIPMENT	17	N	1.000	PC	3GZF101018-74	17	N	1.000	PC	2
21						MMOD_STICKER	930	N	1.000	PC	3GZF101032-15	930	N	1.000	PC	2
22						MMOD_CONN_INTE	390	N	1.000	PC	3GZF101053-1	390	N	1.000	PC	2
23						MMOD_PALLET	750	N	1.000	PC	3GZF374728-26	750	L	1.000	PC	2

Kuva 12. Simuloinnin tuloksena saatu Excel-taulukko.

4.3 Konfiguraattorin säännöt

Konfiguraattorin säännöt määräävät sen, millä perustein mikäkin osa valitaan lopputulosta ajatellen. Konfiguraattorin ensimmäinen tehtävä on validointiprosessi. Se tarkastaa peruskoodin Z-taulusta 13, jonne on listattuna tietty määrä moottorikoodeja. Kenttään syötetty moottorin peruskoodin täytyy täsmätä yhden Z-taulun listassa olevan koodin kanssa, että sen hyväksyminen onnistuu. Tähän tauluun päästään myös käsiksi omalla transaktiolla ja sitä voidaan muokata siellä.

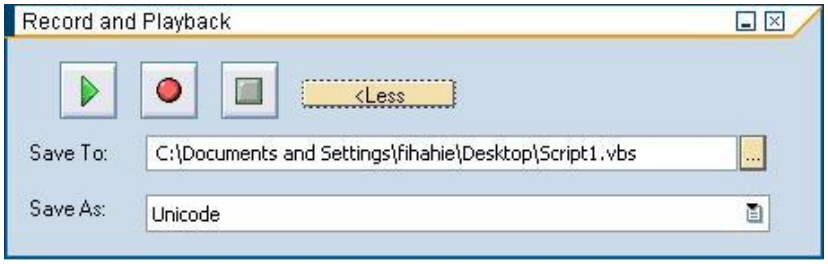


Kuva 13. Z-taulu moottoreiden peruskoodeista.

Samalla tapaa validointiprosessi tapahtuu muissakin kohdissa Z-tauluja hyödyntämällä. Material-kohdassa on kuitenkin vain 2 vaihtoehtoa, 3G ja 3GA, jotka toimivat validointiperusteena. Tämän jälkeen annetut lähtötiedot syötetään ABAP-koodin *CE_C_PROCESSING* FM:lle (Function Module), jossa konfigurointi tapahtuu. Sen sisällä on useampia FM:ja, joita valintaprosessi suorittaa. Näidenkin alla voi olla edelleen aliohjelmia ja tästä syystä simulointi on hidasta. Liitteessä 1 on transaktion luojien TCS Confidentalin tekemä tarkempi tekninen määrittely konfiguraattorin valintaprosessista.

4.4 Komentosarjojen nauhoitus

SAPissa on mahdollisuus nauhoittaa käyttäjän jokainen liike. Se on hyvin kätevä menetelmä ja se tapahtuu Record and Playback –toiminnon avulla (Kuva 14.). Nauhoittaminen on niinkin yksinkertaista, että käyttäjä painaa nauhoituksen päälle, tekee tarvittavat toiminnot ja painaa stop-painiketta. Sen jälkeen käyttäjälle avautuu tiedosto, jossa käyttäjän tekemät toiminnot ovat komentosarjana VBS-kielellä.



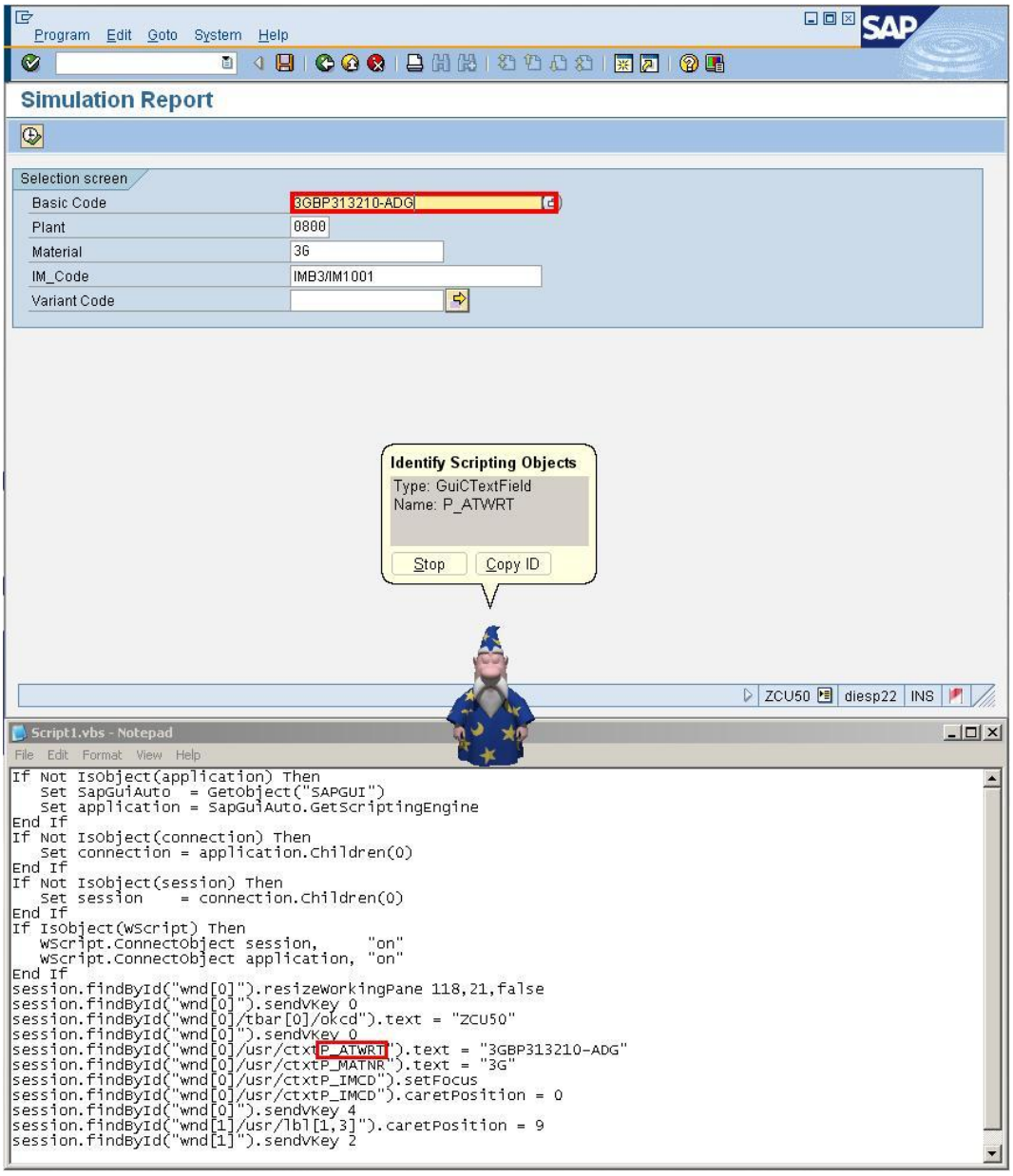
Kuva 14. Nauhuri.

Objekteja ovat mitkä tahansa tekstikentät, valintakentät, ikkunat, napit ja kaikki vastaava mitä käyttöliittymässä näkyy. SAPissa on 2 menetelmää, joilla saadaan selville objektien tunnisteet. Nämä menetelmät ovat äskettäin mainittu komentosarjojen nauhoitus sekä velho, joista molempia käytetään hyväksi tässä projektissa. Näiden hyöty tulee esiin myöhemmin, kun ohjelmoidaan VBA:lla makrokoodi, jonka avulla tulisi saada luotujen rakenteiden ensimmäinen taso ulos SA-Pista. Nauhurin avulla koodin kirjotus tapahtuu itsestään ja velhoa apua käyttäen nähdään visuaalisesti, mikä kohta koodissa tarkoittaa mitäkin toimintoa.

The Scripting Wizardin, eli velhon, saa esiin samasta valikosta kuin komentosarjan nauhoitusominaisuuden. SAPiin ilmestyy velhon näköinen mies, jonka tarkoitus on auttaa käyttäjää graafisesti. Tällä velholla on 2 keskeistä tehtävää, joista ensimmäinen on vianetsintä. Toinen tehtävä sillä on hit test.

Hit testiä käytetään tunnistamaan SAPin graafisessa käyttöliittymässä olevia objektien tunnisteita. Kun käyttäjä liikuttaa hit test -tilassa hiiren osoitinta ruudulla, näyttää velho informaation objektista (Kuva 15.). Käyttäjä voi kopioida velhon kertoman objektin tunnisteen leikepöydälle ja käyttää sitä sen jälkeen omissa komentosarjoissa. Velhoa voidaan siis käyttää apuna komentosarjojen teossa niin kuin nauhoitustakin.

Velho on helposti ymmärrettävä, koska käyttäjä voi visuaalisesti nähdä mikä tunniste on milläkin objektilla. Käyttäjä katsoo velhon kertoman tunnisteen ja voi sen jälkeen etsiä nauhoitetusta koodista tämän kohdan (Kuva 15.). Nauhoituksen ja velhon yhteiskäyttö onkin hyvä apu kokemattomalle ohjelmoijalle. Pelkkä nauhoituksen tekeminen vaatii syvempää SAPin ymmärtämistä.



Kuva 15. Komentosarjojen nauhoitus ja velho yhteiskäytössä.

5 SOVELLUKSEN SUUNNITTELU

Tässä luvussa käydään läpi sovellukselle annettuja vaatimuksia ja analysoidaan niiden toteutuskelpoisuutta. Sen jälkeen suunnitellaan toteutustapoja, joita lähestytään erilaisten kaavioiden myötä. Lisäksi mietitään millaisin eri tekniikoin sovellus tullaan toteuttamaan, mutta yksityiskohtaisiin spesifikaatioihin ei paneuduta.

5.1 Yleiskuvaus

Lähtökohtana on uuden sovellusohjelman suunnittelu. Ohjelman voisi tyypiltään luokitella kaupallis-hallinnollisiin järjestelmiin sekä varus- ja työkaluohjelmistoihin. SAP-ohjelmointi liittyy ensimmäiseen tyyppiin, mutta varsinaista vertailua voidaan pitää työkaluohjelmistona. Ohjelmassa tärkeänä voidaan pitää sen kokoa ja käsiteltävän tiedon määrää yhdistettynä luotettavuuteen.

Ohjelman toteutustapa on vesiputousmallin mukainen tapa, koska lopputulosta on tässä tapauksessa vaikea hahmottaa. Vesiputousmallissa jokaisen välietapin jälkeen tarkastetaan tilanne ja mietitään seuraavaa askelta. Etapit on jaoteltu vaatimusten keräämisvaiheeseen, jota seuraa niiden analyysi ja määritys. Seuraavaksi siirrytään suunnitteluvaiheeseen ja toteutukseen. Viimeiseksi liitetään osat yhteen ja siirrytään käyttöönottoon, johon kuuluu myös kiinteästi ylläpito.

5.2 Esitutkintavaihe ja vaatimusten kerääminen

Esitutkimus oli pitkä prosessi, kun ohjelman toteutuskelpoisuudesta ei oltu varmoja. Ohjelman idea oli kuitenkin hautunut järjestelmäkehitysosaston piirissä noin vuoden verran. Silloin oli vaikea arvioida kannattaisiko sovelluksen kehittämistä jatkaa. Aluksi tarkoituksena olikin tutkia vain mahdollisia ratkaisumalleja ja toteutustapoja. Ohjelma päätettiin kuitenkin toteuttaa ja sen taustalta löytyi järjestelmäkehityksen ulkopuolinen henkilö, joka toteuttaa SAPin koodausta työkseen. Makron koodaaminen oli siis mahdollista ja ohjelmaa pystyttiin ajattelemaan nyt realistisemmin. Tämän jälkeen alkoi vasta todellinen vaatimusten kerääminen.

Motors ei ole ohjelmistotuotantoon perustuva yritys, joten ohjelmien tuottaminen on aina erityslaatuinen tapaus. Ensimmäiset vaatimukset tulivat samaan aikaan kuin itse idea ohjelmasta syntyi. Vaatimukset ovat kertyneet sykleissä ja niitä on kerätty silloin, kun ohjelmaidea on nostettu puheenaiheeksi. Vaatimuksia on kerätty haastattelemalla, sähköpostikyselyillä sekä ryhmäkeskusteluissa.

Lähtökohta on kuitenkin esitystapapohjainen vaatimusten keruu, minkä tarkoitus on kartoittaa vaatimuksia ohjelman tekemisen aikana. Ohjelmalla oli kuitenkin jo aluksi selkeät pääpiirteet, joita tulisi noudattaa ja ne esitetään taulukossa 1 yleisten vaatimusten kohdalla. Ohjelmaa aloitettiin tekemään osa-alue kerrallaan ja eri vaiheiden jälkeen katsottiin, onko toteutustapa hyvä vai huono. Se nosti esiin uusia vaatimuksia. Vaatimusten keräämisestä pidettiin vielä palaveri, jonka pohjalta on laadittu vaatimusluettelo.

Taulukko 1. Vaatimusluettelo.

Vaatimus	Vaatimuksen esittäjä	Päivämäärä	Prioriteetti	Vaatimuksen perustelu
Yleiset vaatimukset				
ZCU50-transaktion automatisoiminen	Järjestelmäkehitys	11/2011	Välttämätön	Suurta määrää on lähes mahdoton toteuttaa ilman automatisointia
Automaattinen moottorirakenteiden haku Teamcenteristä	Järjestelmäkehitys	11/2011	Välttämätön	Suurta määrää on lähes mahdoton toteuttaa ilman automatisointia
Moottoreiden rakennerivien automaattinen vertailu	Järjestelmäkehitys	11/2011	Välttämätön	Tuloksen hyödyntäminen
Automaattinen tulostus Excel-taulukkoon rakennerivit rinnakkain	Järjestelmäkehitys	4/2012	Tärkeä	Vertailun helppous
Sovellus hakee Teamcenter-järjestelmän klassifioinnista rakenteet	Vesa Lappalainen	9/2012	Voidaan toteuttaa myöhemmin	-
Koodin muokkaaminen				
Sovellus tukee laajennettavuutta	Järjestelmäkehitys	Palaveri 29.10.2012	Välttämätön	Sovelluksen käyttäminen jatkossa
Ohjelmaa voidaan muokata toimimaan muissakin Teamceterin ja SAPin välisissä vertailuissa	JP Alanen	Palaveri 29.10.2012	Tärkeä	Sovelluksen käyttäminen jatkossa

Ohjelmakoodin jokainen osa-alue tulee olla kommentoituna tarkasti	Sinikka Saunaaho	Palaveri 29.10.2012	Välttämätön	Sovelluksen käyttäminen jatkossa
Käytettävyys				
Käyttöliittymä on kevyt rakenteeltaan ja selkeä ulkoasultaan	Järjestelmäkehitys	Palaveri 29.10.2012	Tärkeä	Ohjelmaa tulee käyttämään projektin ulkopuoliset henkilöt
Käyttöliittymän tulee tarjota käyttöohje ohjelman toiminnasta	Järjestelmäkehitys	10/2012	Tärkeä	Ohjelmaa tulee käyttämään projektin ulkopuoliset henkilöt
Käyttöliittymän tulee olla englannin kielellä	Järjestelmäkehitys	10/2012	Tärkeä	Sovelluksen mahdollinen käyttö Kiinan tehtaalla
Lopputuloksessa virheelliset rakennerivit ovat värjätty selvästi erottuvaksi	Järjestelmäkehitys	9/2012	Välttämätön	Lopputuloksen helppo löytäminen suuresta massasta

5.3 Vaatimusten analysointi

Ohjelman toiminnan kannalta analysointivaihe on tärkeä, koska pienikin virhe voi vaikuttaa koko ohjelman toimintaan. Vaatimusten kerääminen tuotti välttämättömimmät ominaisuudet, jotka ohjelman tulee täyttää. Näitä ominaisuuksia kutsutaan eksplisiittisiksi vaatimuksiksi. On olemassa myös vaatimuksia, joita oletetaan ohjelman toiminnalta ilman erillistä määrittelyä. Tällaisia ominaisuuksia ovat yleensä hyvä ylläpidettävyys, virheettömyys ja käyttäjäläheisyys. Näitä vaatimuksia kutsutaan yleisesti implisiittisiksi vaatimuksiksi. Nämä on otettava huomioon ohjelman toteutuksessa vaikka vaatimusten keruu ei sitä suoranaisesti ilmaise.

Kaikissa vaatimuksissa on ajateltu ohjelman kohderyhmiä. Pääkohderyhmänä on suunnitteluosasto, mutta myös järjestelmäkehitysosasto tai kuka tahansa muu Teamcenterin ja SAPin käyttäjä voi hyötyä ohjelmasta. Ohjelmaa on ajateltu käytettäväksi esimerkiksi puolen vuoden välein, jolloin vertailtaisiin kaikkia simuloitavia moottoreita yhdellä kertaa. Yksittäisiä moottoreita voidaan tarkistaa useammin.

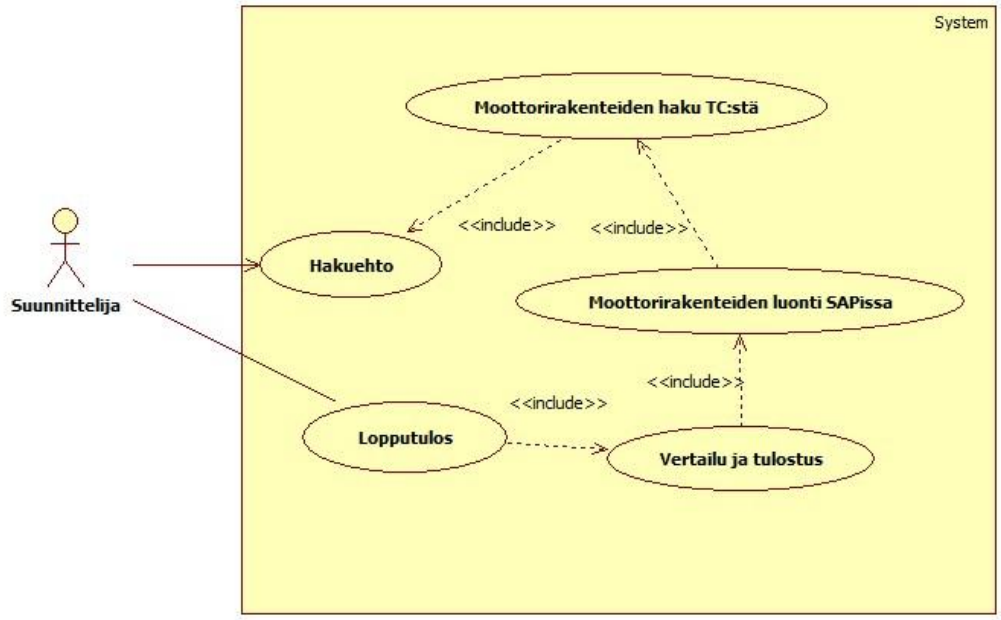
Vaatimusluettelosta välttämättömät ja tärkeät ominaisuudet voidaan nähdä toteutuskelpoisiksi. Vaatimusluettelo ei ole täydellinen, mutta olettamuksena on, että sovelluksesta tulee Windows-pohjainen. Järjestelmävaatimuksiakaan ei mainittu

vaatimusten keräämisessä, mutta loppukäyttäjältä oletetaan Windows-käyttöjärjestelmää sekä Teamcenterin ja SAPin asennusta ABB:n ympäristössä.

Analysoitaessa vaatimuksia huomattiin, että rakennerivit joudutaan rajoittamaan nimikkeen, osanumeron ja kappalemäärän vertailuun. Tämä johtui Teamcenterin ja SAPin erilaisesta ulosannista. Aivan täydellistä rakennerivien vertailua ei voida toteuttaa, mutta toisaalta korjaukset tehtäisiinkin nimikkeen ja osanumeron perusteella.

5.4 Käyttötapaukset

Käyttötapauksia kuvataan usein kaavioin. Kuvan 16 tarkoitus on tehdä mallinnus käyttötapauksesta, joka vastaa kysymyksiin, mihin tapauksiin sovellus on tarkoitettu, kuka tai ketkä ovat käyttäjiä ja mitkä ovat sovelluksen oleellisimmat toiminnot. Suunnittelijan tehtävänä on tehdä hakuehto ja tarkastaa tulos. Sovellus on riippuvainen käyttäjän hakuehdosta ja sen tehtävänä on käynnistää kaikki toiminnot. Toiminnot riippuvat toisistaan, jokainen edellisestä toiminnosta.



Kuva 16. Käyttötapauskaavio.

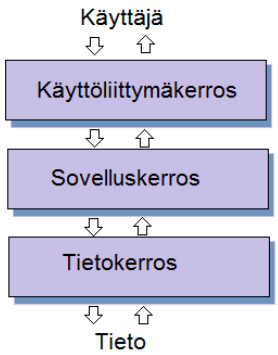
5.5 Tekninen määrittely

Tekninen määrittely kuvaa tekniikoita, joita valitaan ohjelman toteuttamista varten. Sovellus tullaan toteuttamaan pääasiassa Java-ohjelmointikielellä hyödyntäen omia vahvuusalueita. Yksinkertainen graafinen käyttöliittymä tullaan myös toteuttamaan Javalla, käyttäen hyödyksi Java Swing-kirjastoja. Moottorihaussa Teamcenterin tietokannassa tullaan käyttämään SQL-kyselykieltä, mutta Javan sisäisesti. Tässä tarvitaan Microsoft SQL:n omia kirjastoja, jotta voidaan Javalla yhdistää Teamcenterin palvelimelle.

Makron ohjelmoinnissa hyödynnetään SAPin nauhoitusominaisuutta sekä Excelissä olevaa vastaavanlaista ominaisuutta. Makro tullaan toteuttamaan VBA-ohjelmointikielellä ja sen tulostus tapahtuu Exceliin. VBA on valittu siksi, että makron ohjelmoimiseen on tarvittaessa apua saatavilla. Vertailu tulee tapahtumaan Javalla hyödyntäen hajautustauluja, joihin moottorit ja rakenteet tullaan sijoittamaan. Rakenteiden tulostus tehdään Excel-taulukkoon myös Javalla. Tämä vaatii Java Excel APIn tai Apache POI:n asennuksen, jotka sisältävät kirjastoja Microsoftin tuotteiden hyödyntämisestä Java-ohjelmoinnissa.

5.6 Arkkitehtuuri

Sovellus on tarkoitus toteuttaa kolmitasomallin mukaisesti. Se on jaettu kolmeen eri tasoon, jossa kullakin tasolla on omat tehtävänsä. Tarkoituksena olisi, että jokainen kerros toimisi itsenäisesti, mutta osaisi kommunikoida lähimpien kerrosten kanssa (Kuva 17.).



Kuva 17. Kolmitasomalli.

Kuvassa alimpana on tieto, joka kuvaa tässä tapauksessa moottorirakenteita Teamcenterin tietokannassa. Tietokerroksen tarkoitus on hallita tietoa ja hakea data tietokannasta. SAPin puolella ei varsinaisesti ole suoraa tietoa, mutta tietokerroksen tehtävänä on hakea moottorirakenteet luomalla ne siellä. Tietokerros toimii rajapintana tiedon ja sovelluskerroksen välillä.

Keskimmäisenä on sovelluskerros, jonka tehtävä on ohjata ohjelman toimintaa. Tässä kerroksessa on ohjelman äly ja logiikka. Sovelluskerroksen tarkoitus on pyytää, vastaanottaa, muokata ja hyödyntää tietoa. Se yhdistää käyttöliittymä- ja tietokerroksen itsensä kautta eli toimii kommunikaation keskipisteenä.

Ylin kerros on käyttöliittymäkerros, joka toteutetaan viimeisenä. Sovellus toimii ilman tätäkin kerrosta. Käyttöliittymä toimii rajapintana käyttäjän ja sovelluskerroksen välillä. Käyttöliittymälle syötetään alkuehdot, mutta se myös näyttää ohjelman edistymisen ja lopputuloksen.

Koko tapahtumaketju lähtee siis käyttäjästä, joka päättää mitä moottoreita haetaan. Käyttäjä tekee päätöksen käyttöliittymäkerroksessa. Tästä siirrytään sovelluskerroksen kautta tietokerrokseen, josta haetaan käyttäjän pyytämä data. Data palautuu sovelluskerrokseen, jossa suoritetaan datalle haluttuja toimenpiteitä. Muokattu data siirretään sovelluskerrokselta käyttöliittymäkerrokseen, jossa käyttäjä voi nähdä lopputuloksen.

6 SOVELLUKSESSA KÄYTETYT TEKNIIKAT

Tässä luvussa esitellään ohjelmointikielet, joita sovelluksessa on käytetty. Nämä ohjelmointikielet ovat Java ja VBA. Tämä luku käsittää myös sen, miksi nämä kielet on valittu sekä kertoo niiden synnystä ja ominaisuuksista. Sen lisäksi ohjelmassa käytettiin yhtä kyselykieltä nimeltä SQL, jolla päästään käsiksi Teamcenterin tietokantaan.

6.1 Java-ohjelmointikieli

Tämän tutkimuksen sovellusohjelma on pääsääntöisesti tehty ohjelmointikieli Javaa hyödyntäen. Se on valittu siksi, koska Java on monipuolinen, tehokas, turvallinen ja sen alustan laaja siirrettävyys tekee siitä ihanteellisen tekniikan tähänkin projektiin. Javaan törmää kaikkialla, kuten tavallisissa matkapuhelimissa ja pelikonsoleissa, mutta se on käytössä myös esimerkiksi mullistavissa tietokeskuksissa ja tieteen kehitykseen tarvittavissa supertietokoneissa. /8/

6.1.1 Synty ja ajatusmaailma

Javan kehitys on saanut alkunsa kesäkuussa vuonna 1991, kun Sun Microsystemsin James Gosling aloitti kollegoidensa kanssa projektin nimeltä *Oak*. Sen ensimmäinen julkinen versio Java 1.0 tuli markkinoille vuonna 1995. Sen suosio kasvoi, kun suurimmat selaimet sisällyttivät Javan osaksi heidän vakiokokoonpanoa Java Appletiksi. /7/ Oracle Corporation osti Sun Microsystemsin vuonna 2009 ja vastaa nykypäivänä Javan kehityksestä. /13/

Javan syntaksi muistuttaa läheisesti C ja C++ kieliä. Sen ohjelmointiparadigmana eli ajatustapana on yksinkertainen oliopohjainen ajattelu. Java sekoitetaan usein myös JavaScriptiin nimensä ja samantapaisen syntaksin takia, mutta ovat todellisuudessa vain kaukaista sukua toisilleen. Oliopohjainen ajattelutapa oli yks peruspilari, kun Javaa lähdettiin alunperin luomaan. Toinen ajatus oli, että sama ohjelma voitaisiin suorittaa useassa eri käyttöjärjestelmässä. Kolmantena sen täytyi sisältää sisäänrakennettu tuki tietoverkkojen käytössä. Neljäs tavoite projektilla oli saada koodi ajettua turvallisesti etäkäytössä. Viimeinen tavoite oli sen helppokäyt-

töisyys ja hyvien ominaisuuksien poimiminen muista oliopohjaisista ohjelmointikielistä. /7/

6.1.2 Olio-ohjelmointi

Olio-ohjelmointi tarkoittaa tietyntapaista ohjelmointimenetelmää ja ohjelmointikielen rakennetta. Sille ominaista on, että data ja koodi yhdistetään kokonaisuuksiksi, joita kutsutaan olioiksi. Yhdelle oliolle on roolitettu tietty tehtävä ohjelmassa. Se on siis itsenäinen osa ohjelmakoodissa, joka vaikuttaa ohjelman käyttäytymiseen. Oliot voivat käsitellä tietoa ja välittää sitä muille olioille. Oliolla voi myös olla itsenäisiä ominaisuuksia, joita kutsutaan olion ilmentymiksi. /7/

Oliopohjaisen ohjelmoinnin hyötyjä ovat sen organisoiminen pienempiin palasiin. Nämä pienet palaset ovat sitten helposti hallittavissa ja niiden toimintoja pystytään tarkasti määrittämään. Toinen hyöty sillä on koodin käytettävyys uudelleen. Tällöin tiettyä koodia ei tarvitse kirjoittaa toistamiseen. Oliopohjaista koodia on myös helppo ylläpitää tai laajentaa. Jos ohjelmassa muutetaan jokin kohta, se ei vaikuta kaikkialle. /7/ Laajentaminen on helppoa rakentamalla koodia luokka kerrallaan. Luokka määrittelee olion ominaisuudet.

6.1.3 Ohjelmointiympäristö

Java Runtime Environment (JRE) on ohjelman ajamiseen tarvittava ajoympäristö. Tämä ajoympäristö on osa Javan kehitysympäristöä Java Development Kitiä (JDK). Tässä työssä on käytetty JDK:n ohjelmistopakettia Java Enterprise Editionia (Java EE). /12/ Se saatiin käyttöön Eclipse-ohjelmointiympäristön asennuspaketista, joka piti sisällään Java EE:n. Eclipse on tuttu ohjelmisto kouluympäristöstä ja se on käytössä myös Vaasan ammattikorkeakoulussa.

Eclipse on ilmainen, avoimeen lähdekoodiin perustuva ohjelmointiympäristö, joka julkistettiin vuonna 2001. Se tukee pääsääntöisesti Java-ohjelmointikieltä ja se käyttää Javan ohjelmointirajapintaa. /15/ Eclipse toimii sekä editorina että kääntäjänä. Editorin ominaisuuksia ovat selkeyttävät värijaot syntaksissa. Se osaa myös muotoilla koodia ja ehdottaa seuraavaa toimintoa käyttäjän kirjottaessa vasta sanan alkua. Objekteja luodessa se näyttää myös ohjeet sen käyttömahdollisuuksis-

ta, jotka ovat vastaavanlaiset mitä Oraclen web-sivustolta löydetään. Kääntäjällä on ajon lisäksi ominaisuus, joka etsii syntaksivirheitä koodista ja löytää sen lisäksi oikeat virhekohdat. Se ehdottaa myös korjaustapoja, joilla näitä virheitä ei tulisi. Eclipse sisältää myös yksinomaan virheen jäljittämiseen perustuvan debuggerin.

6.2 VBA-ohjelmointikieli

Ensimmäiset vaiheet Visual Basicin syntymisestä oli jo vuodelta 1964, jolloin BASIC-niminen ohjelmointikieli julkaistiin. Sen ohjelmointiparadigmana on yleinen imperatiivinen ohjelmointitapa, jonka idea perustuu aliohjelmien käyttöön. Ensimmäinen virallinen Visual Basic -ohjelmointikieli julkaistiin vuonna 1991 ja sen kehittäjänä toimi Microsoft Corporation. Sen kehityksen tarkoituksena oli nopeuttaa ja helpottaa ohjelmointia uudessa graafisessa käyttöjärjestelmässä. /6/

Visual Basicistä on tehty 6 versiota. Vuonna 2002 Microsoft julkaisi täysin uudelleen kirjoitetun ohjelmointikielen Visual Basic .NETin. Se oli täysin olio-ohjelmointiin perustuva, vaikka VB6-versio oli pitkälti sitä myös. VBA-ohjelmointikieli on luotu Visual Basic 6:n pohjalta ja räätälöity erityisvaatimuksiin. VBA:n syntaksi muistuttaa lähes täysin VB6:n syntaksia. VBA eli Visual Basic for Applications on luotu Mircosoft Officen ohjelmien, kuten Excelin ja Wordin laajentamiseen ja automatisoimiseen. /6/

VBA-ohjelmointikieli toimii Microsoft Office –ohjelmien makrokielenä. ”Makro on komentojen ja ohjeiden sarja, jotka yhdistetään yhdeksi komennoksi tehtävän automaattista suorittamista varten”. Makroja voidaan ohjelmoida itse, mutta VBA mahdollistaa nauhoitusominaisuuden, joka kirjoittaa ohjelmointikieltä käyttäjän toimintojen mukaan. Suurin hyöty makrosta saadaan kuitenkin kirjoittamalla koodia itse. /11/

Makron koodaamisesta voidaan hyötyä, jos halutaan nopeuttaa rutiiniluontoista muokkausta tai muotoilua. Toisena makron käytön taustalla voisi olla monen komennon yhdistäminen yhdeksi. Sitä voidaan käyttää helpottamaan valintaikkunoiden vaihtoehdoissa tai sillä voidaan automatisoida monimutkaiset tehtäväsarjat. /11/

6.3 SQL-kyselykieli

SQL eli Structured Query Language tarkoittaa rakenteista kyselykieltä. Sen avulla voidaan hallita reaalitietokantoja. Sen varhaisimmat kehitysvaiheet ovat 1970-luvun lopulta, kun teknologiayritys IBM kehitti kielen DB2:ta varten. Sen hurja potentiaali huomattiin ja SQL:sta on tullut lähes standardinomainen tietokantojen ylläpidossa. /1/

SQL ei ole kuitenkaan rakenteellinen kieli, joten se määrittelee ennemminkin mitä tietoa haetaan kuin miten sitä haetaan. Sen avulla voidaan muokata rakennetta tietokannassa, päivittää ja tehdä kyselyjä tietokantaan. Sillä voidaan myös muuttaa turva-asetuksia tai lisätä käyttäjien oikeuksia järjestelmässä. SQL toteutuksia on useita erilaisia. Eri toteutuksilla on tarkoitus helpottaa tietokannan hallintaa ja niihin on tehty erilaisia graafisia käyttöliittymiä. /1/

Microsoft on toteuttanut monia erilaisia SQL-työkaluja ja ohjelmia. Tässä työssä on käytetty Microsoftin MS SQL Serveriä tietokanta-alustana. /1/ SQL:n rooli tässä työssä on vähäinen, mutta erittäin merkittävä työn kannalta. Se liitetään osana Javalla tehtyyn ohjelmointikoodiin ja sen avulla saadaan luettua Teamcenterin tietokannasta moottorikoodit ja rakennerivit.

7 KÄYTTÖLIITTYMÄN SUUNNITTELU

Käyttöliittymä toteutetaan graafisena ohjelmana. Sen toteutus tulee tapahtumaan Java-ohjelmointikielellä ilman käyttöliittymien rakentamiseen erikoistuneita ohjelmia. Tällä käyttöliittymällä ei tule olemaan kustannuksia. Sitä tullaan testaamaan ABB:n ympäristössä Windows XP- ja Windows 7 -käyttöjärjestelmillä. Muilla laitteistoilla sen ei luultavasti tarvitsekaan toimia. Sen suunnitteluun liittyy kuitenkin useita vaiheita ja tärkein kriteeri on sen ajattelu loppukäyttäjän kannalta. Sovellusta tullaan käyttämään puolen vuoden välein, joten sen tehokkuus ja erityten opittavuus tulisi mennä perille jo ensimmäisellä käyttökerralla.

Sovelluksen käyttötarkoitus on toimia suunnittelijoiden työkaluna. Suunnittelijat voitaisiin lukea edistyneempiin käyttäjiin, joten käyttöliittymän toteutus voidaan toteuttaa samoilla lähtötiedoilla, jotka koskisivat itseäni. Heille termistö on tuttua ja tietokoneen käyttö päivittäistä. Ohjelmasta ei tarvitse laatia suurempaa manuaalia, vaan pienet ohjeet sen toiminnoista riittävät.

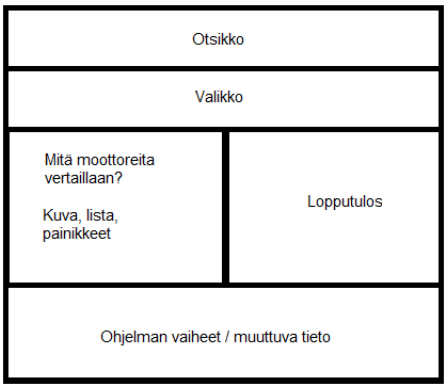
Käyttöliittymän on tarkoitus olla standardisoitu, jotta käyttäjän olisi mahdollisimman helppo oppia sen käyttö. Standardisoidulla tarkoitetaan tässä sitä, että esimerkiksi File-valikko löytyy ylhäältä vasemmalta, niin kuin useimmissa ohjelmissa. Mitään uutta käyttötapaa ei tule olemaan, vaan käyttäjän perustiedoilla pystytään ohjelmaa käyttämään.

7.1 Visuaalinen suunnittelu

Visuaalinen kuva käyttöliittymän layoutista muodostui hyvin nopeasti. Värimaailman täytyi muistuttaa tavallista Windows-ohjelman harmaasävyistä tyyliä. Ajatuksena oli myös, että käyttöliittymä on mahdollisimman pienikokoinen fyysiseltä kooltaan eli pikselitasoltaan. Tässä oli tarkoituksena se, että huomataan SAPin pyörivän taustalla ja voidaan seurata samanaikaisesti ohjelman statusta.

Jäsentelyn lähtökohtina oli saada moottorin haku, tulos ja tieto ohjelman etenemisestä näkymään samanaikaisesti ruudulle. Lisäksi ohjelmaan oli tarkoitus jättää laajentamisen mahdollisuus. Alkuperäinen jäsentely nähdään kuvasta 18. Se on jäsennelty samanlaisella tyylillä kuin useampi käyttöliittymä. Keskimmäiset ruu-

dut, joissa ovat moottorihaku ja lopputulos, vievät layoutilla suurimman tilan. Tämä keskiosa voidaan jakaa vielä kolmeen osaan, jos ohjelmaa laajennetaan esimerkiksi uudella hakumenetelmällä.

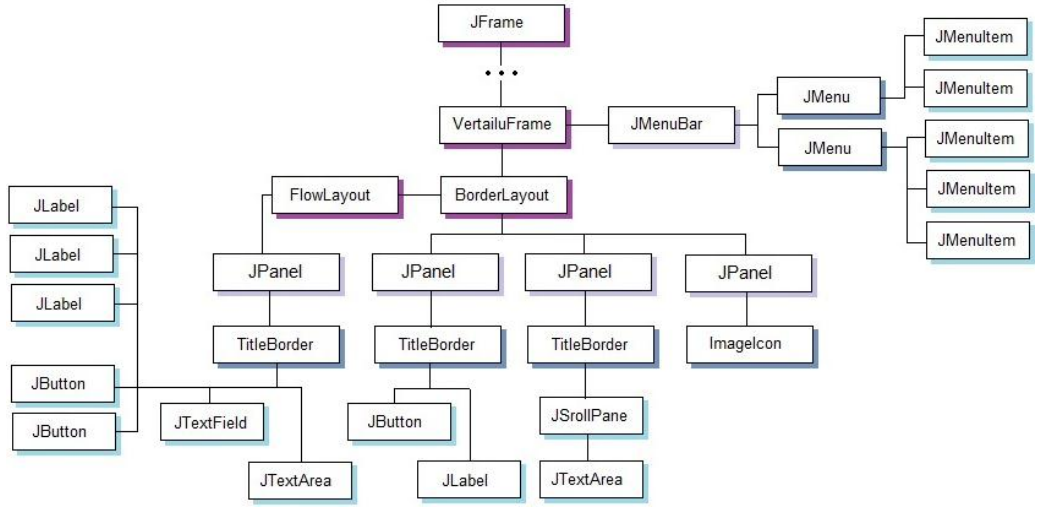


Kuva 18. Rakenteen jäsentely.

7.2 Toiminnallinen suunnittelu

Käyttöliittymän toiminnallinen suunnittelu tapahtui ohjelmakoodin kirjoittamisen yhteydessä. Se tapahtui jäsentelyn perusteella, mitä yritettiin noudattaa mahdollisimman tarkasti. Vaikeuksia tuotti eniten ruudunjako näytöllä. Tässä nähtiin parhaaksi, että skaalausmahdollisuus otetaan pois käytöstä ja ruutu pysyy aina samankokoisena. Toteutus tapahtui hyödyntäen Javan luokkaa, joka periytyy käyttöliittymien tekemiseen tarkoitetusta JFrame-luokasta.

Kuvassa 19 nähdään käyttöliittymän rakenteellinen suunnittelu toteutustavasta. Kuvion objektit on nimetty Javan Swing-kirjaston mukaan. Harmaataustaiset ruudut, joita on kuvassa 5 kappaletta, ovat ensimmäinen näkyvä taso ruudulla. Ensimmäinen näistä on oikealla sijaitseva valikko, joka muodostuu sen sisäisistä komponenteista. Muut 4 ovat ohjelman osioita ruudulla, joita ohjaa jokaisen oma layoutin käsittelijä. Jokaisella osiolla on oma sisältönsä komponentteineen.



Kuva 19. Käyttöliittymän rakenne Java-ympäristössä.

8 SOVELLUKSEN TOTEUTUS

Työkalu on toteutettu Javalla ja VBA:lla, joista jälkimmäisen osuus rajautuu pelkästään simulointiosuuteen. Lisäksi työkalu käyttää hyödyksi SQL-kyselyitä ja tavallista Visua Basicia. Java-koodi on jaettu 9:ään eri luokkaan, jotka ovat:

- graafinen käyttöliittymä
- rakennerivien hakeminen TC:stä
- rakennerivien hakeminen SAPista
- rakennerivien vertailu ja lopputuloksen kirjoittaminen
- prosessien kuuntelu
- väliaikaistiedostojen poisto
- valikkorivi
- ohjeet.

8.1 Pääluokka ja graafinen käyttöliittymä

Käyttöliittymä on integroitu pääluokkaan. Pääluokan tarkoitus on olla mahdollisimman pieni ja sen ainoana tehtävänä on toimia muiden luokkien kommunikaation keskipisteenä. Pääluokka periytyy JFrame-luokasta ja implementoituu ActionListener-rajapinnasta. JFrame mahdollistaa kehyksien luonnin. ActionListenerin nimestä voidaankin jo päätellä, että se mahdollistaa kuuntelun käyttäjän toiminnoista sovellusikkunassa.

Pääluokka sisältää myös konstruktorin, jossa luodaan sovellusikkunan sisältö. Lisäksi pääohjelmalla on metodeja, joissa kuunnellaan eri painikkeiden painamista. Sillä on myös metodi, jonka avulla voidaan lisätä moottorikoodeja hakulistaan. Tärkeimpänä voidaan kuitenkin pitää metodia, joka käy kokonaisuudessaan ohjelman läpi.

8.1.1 Main-metodi

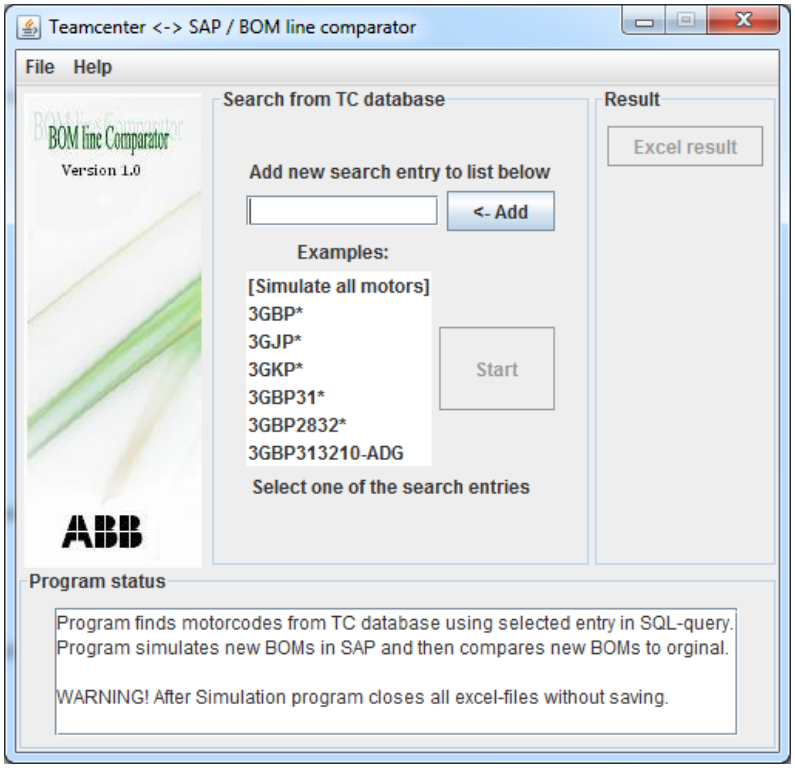
Pääluokassa on koko ohjelman ainut Main-metodi. Tämä on käytännössä kaiken pääohjelma. Ohjelmaa ajettaessa se käy ainoastaan tämän metodin läpi. Metodissa määritetään sovellusikkunalle koko ja sijainti sekä asetetaan sovellusikkuna näky-

väksi. Sitten se saa vielä oman luokkansa konstruktorin sisällön kenttineen ja painikkeineen. Tämän lisäksi se vielä kutsuu toisesta luokasta valikkoriviä saadakseen sen käyttöönsä.

8.1.2 Käyttöliittymän sisältö

Käyttöliittymän sisällöstä vastaa konstruktori, jossa aluksi jaetaan sovellusikkuna pienempiin osiin. Sen ulkoasu koostuu 4 osasta, jossa vasempaan reunaan on tehty kuva ulkoasun piristykseksi. Keskimmäisessä osiossa on lista hakuehdoista ja käynnistys-painike. Lisäksi siinä on tekstikenttä, jonka avulla hakulistaan voidaan lisätä omia hakuja. Oikeassa reunassa on nappi, joka avaa lopputuloksen ohjelman suorituksen jälkeen. Alhaalla on statuskenttä, jonka tarkoitus on informoida ohjelman ajonaikaisista tapahtumista.

Konstruktorissa luodaan painikkeet, jotka lisätään edellisiin osioihin. Siellä luodaan myös tekstikenttä, tekstinsyöttökenttä ja lista. Sen lisäksi siellä lisätään listaan valmiit hakuehdot. Hakuehdoille luodaan oma listan kuuntelija, jonka avulla ohjelma osaa valita oikean hakuehdon ohjelman käynnistyessä. Statusalueen tekstikenttään luodaan vierityspalkki automaattiseksi siten, että tekstin tulostuessa se vierii itsestään uusimman tekstin kohdalle. Kuvasta 20 nähdään millainen käyttöliittymä aikaansaatiin.



Kuva 20. Sovellusikkuna.

8.1.3 Ohjelman suoritus

Ohjelman suoritus on omassa metodissaan ja se käynnistyy, kun hakuehto on valittu ja start-painiketta painettu. Se luo oliot kaikista muista luokista saadakseen niiden metodit käyttöönsä. Se ajaa halutussa järjestyksessä ohjelman läpi alkaen Teamcenterin moottorihausta ja päättyen Exceliin kirjoitettuun lopputulokseen. Se kirjoittaa myös ohjelman etenemisen vaiheista tietoa ja tulostaa sen käyttöliittymän statuskenttään. Jokaisesta vaiheen muutoksesta kirjoitetaan myös kellonaika, joka tulee omasta metodistaan.

8.2 Rakennerivien hakeminen TC:stä

Rakennerivien hakeminen TC:stä on oma luokkansa. Ohjelmaa ajettaessa ensimmäinen olio periytyy tästä luokasta. Tämän luokan metodeja kutsutaan ensimmäisenä siksi, että väärät rakennerivit ovat juuri Teamcenterissä. Tarvitaan tiedo siitä, mitä moottoreita Teamcenterissä on ja mitä moottorirakenteita korjataan. Ohjelmakoodin toimiminen tietokannassa vaatii sen yhdistämiseen jonkun liittymän.

Moottorikoodien ja rakenteiden hakuun tarvitaan SQL-kyselyjä ja niiden järjestyksen ylläpitämiseksi tarvitaan taulukko.

Teamcenterin palvelimelle yhdistämistä varten joudutaan asentamaan Microsoft SQL Server JDBC Driver 3.0 -ajuri, koska se ei ole osa Java SDK:ta. Se tarjoaa yhteyden SQL-serverille JDBC-standardin sovellusohjelman liittymällä. Tämän asennuspaketin sisältä löytyi 2 jar-tiedostoa. Kun nämä lisättiin projektin kirjastokantaan, pystyttiin ottamaan sen kirjastot käyttöön.

8.2.1 Yhdistäminen tietokantaan

Luokka koostuu 3:sta eri metodista, joista ensimmäinen on Teamcenterin tietokantaan yhdistäminen. Käyttääkseen sqljdbc.jar-kirjastoa, sovelluksen täytyy ensin rekisteröidä ajuri. Se tapahtuu alla olevan kuvan 21 ylimmän lauseen tavalla. Yhdistäminen on toteutettu niin, että merkkijonomuuttujalle syötetään palvelimen osoite, tietokannan nimi, käyttäjätunnus ja salasana. Kun kyseinen merkkijono syötetään DriverManager-luokan getConnection-metodille, saadaan yhteys toteutettua palvelimelle.

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
String connectionUrl = "jdbc:sqlserver://fivaa-s-te00101;" +
"databaseName=FIVAAPROD;user=[redacted];password=[redacted]";
conn = DriverManager.getConnection(connectionUrl);
```

Kuva 21. Yhdistäminen Teamcenterin tietokantaan.

8.2.2 Moottorikoodit

Toinen metodi on moottorikoodit ja nimensä mukaan sen tehtävänä on hakea Teamcenterin tietokannasta SQL-lauseella kaikki vertailtavat moottorikoodit. Sen lisäksi sen täytyy siivota turhat moottorikoodit, joita ovat esimerkiksi testikoodit ja virheelliset koodit. Pääluokasta saadaan käyttäjän syöttämä hakuehto, joka voi olla joko yksi moottori tai esimerkiksi jokainen 3GBP-alkuinen moottori.

Tämä hakuehto lisätään merkkijonomuuttujaan, joka syötetään edelleen SQL-lauseeseen. SQL-lause muodostetaan yhdistetyllä merkkijonolla *SELECT * FROM PITEM where PITEM.pitem_id like 'X'*. Tässä X:n paikalle laitetaan ha-

kuehto. Tämän lauseen tarkoitus on hakea PITEM eli osien taulusta sen perusteella kaikki löytyvä, kun osan tunnuksena toimii haluttu hakuehto.

Haun tulos pitää saada järkevästi ylös ja edelleen käytettäväksi. Laittamalla moottorikoodit listaan, saadaan ne myös kätevästi sieltä ulos. Tässä listarakenteena toimii ArrayList, johon voidaan listata ennalta määrittelemätön määrä moottorikoodeja. Moottorikoodit siivotaan pintapuolisesti siten, että jos koodissa on useampi väliviiva kuin yksi, sitä ei tule korjattuun ArrayListaan.

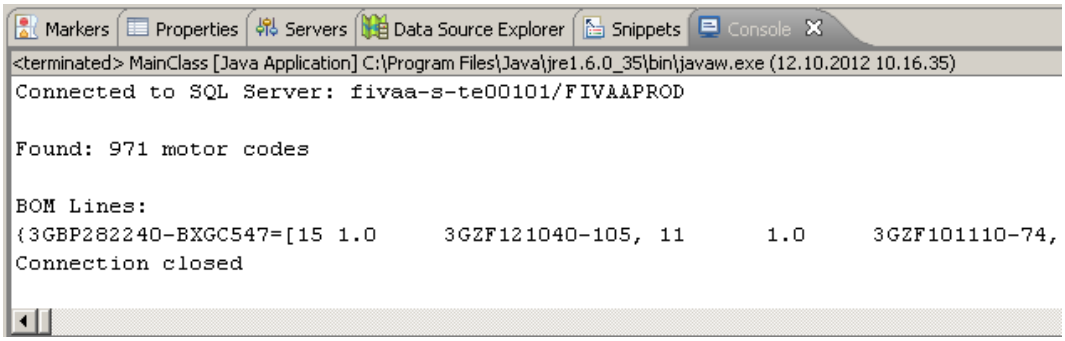
8.2.3 Rakennerivit

Kolmas metodi käsittelee rakennerivejä. Se on käytännössä samanlainen metodi kuin edellinen. Tässä metodissa vain SQL-lause on huomattavasti pitempi, koska rakennerivit ovat vaikeammin saatavilla tietokannasta. Toisin kuin moottorikoodien haussa, SQL-kyselyjä joudutaan tekemään 2 erilaista. Tämä siksi, että jos hakuehtona on ollut vain 1 moottori, kysely on erilainen verrattuna useammaan moottorin kyselyyn. Useamman moottorin kyselyssä joudutaan muodostamaan alkulauseen lisäksi niin monta or-lausetta kuin moottoreita listasta löytyy.

Jotta voidaan hyödyntää tätä hakua, täytyy tulokset saada ylös. Ensimmäisessä haussa saadaan moottorikoodit ylös ja jälkimmäisessä rakenteet kyseiselle moottorille. Luodaan hajautustaulu tyyppiä HashMap, jolle syötetään avaimeksi moottorikoodi. Avaimen arvopariksi lisätään lista moottorirakenteista käyttäen tässäkin ArrayListaa. Listassa yhdellä rivillä on aina 1 rakennerivi, jossa on sekvenssinumero, kappalemäärä ja osan nimi erotettu tabulaattorilla toisistaan. Haku toimii myös siten, että moottorikoodit, jotka eivät sisällä Teamcenter järjestelmässä rakenteita, eivät pääse enää tähän HashMapiin.

Kuvasta 22 voidaan nähdä, miten rakennerivit ovat käytännössä hajautustaulussa. Aaltosulut avaavat HashMapin ja sen ensimmäinen alkio on 3GBP-alkuinen moottorikoodi. Tämä moottorikoodi on siis yksi avain hajautustaulussa. Sitten näemme yhtä kuin –merkin, jonka oikea puoli toimii avaimen arvoparina. Arvoparina on nyt lista, jonka avaa hakasulku. Listan ensimmäisellä rivillä on osanumero, kappalemäärä ja osan tuotekoodi. Tämän listan rivit on erotettu pilkulla, joten

huomataan, että kuviosta näkyy vielä toinenkin rivi. Näitä rivejä eli rakenteita on yhteensä noin 20 kappaletta yhtä moottoria kohden. Tämän hajautustaulun ulossaaminen onkin koko luokan tarkoitus ohjelmassa. Sille on luotu oma metodi, jonka ainoa tehtävä on palauttaa valmis HashMap pääohjelman käyttöön. Se jää sitten odottamaan kutsua ja SAPista saatujen moottorirakenteiden vertailua.



Kuva 22. Eclipsen konsoli, jossa näkyy moottorikoodi ja 2 sen rakenneriviä.

Ongelmana on vielä saada nämä haulla löydetyt moottorikoodit makrokoodin käyttöön SAPin puolella. Koska merkkijonomuuttujaa ei toisesta ohjelmakoodista toiseen voi siirtää, tulostetaan ne väliaikaiseen tekstitiedostoon. Tässä joudutaan tekemään tiedoston kirjoituksen alustus. Sitten iteroidaan HashMap kerran läpi ja tulostetaan sen jokainen avain tiedostoon ilman arvoparia. Tiedostossa 1 avain on yhdellä rivillä.

SAPin transaktio ZCU50 vaatii hakukentässään lisäksi IM-koodin. IM-koodin hakeminen Teamcenterin tietokannasta ja moottoreiden klassifioinnista ei ole enää kovin yksinkertaista. Tähän parhaaksi ratkaisuksi löytyi niiden hakeminen manuaalisesti klassifioinnista. Klassifiointitiedoista kaivettiin esiin ensin jokainen moottori, joka on klassifioitu. Niitä löytyi yhteensä reilu 1200 kappaletta. Sen jälkeen sieltä poimittiin moottorikoodi ja sitä vastaava IM-koodi. Nämä kopioitiin käsin tekstitiedostoon ohjelman käytettäväksi.

Moottorikoodin kirjoittamisen lisäksi se vaatii tekstitiedostoon siis klassifiointitiedostosta moottorikoodia vastaavan IM-koodin. Väliaikainen tekstitiedosto muodostuu lopulta muotoon moottorikoodi, erotusteksti sekä IM koodi. Erotustekstinä käytetään ”IM:” -tekstiä ja sen tarkoitus on makrokoodissa erottaa moot-

torikoodi IM-koodista. Kaikki edelliset kirjoitetaan ilman välimerkkejä ja yksi rivi väliaikaisessa tekstitiedostossa voisi esimerkiksi näyttää tältä: *3GBP313210-ADGIM:IMB3/IM1001*.

8.3 Automatisoitu simulointi

Automatisoitu simulointi on toteutettu VBA:lla. Toteutuksessa on kuitenkin käytetty SAPissa olevaa nauhuria, jolla on saatu transaktion objektien tunnisteet. SAPin nauhuri kirjoittaa VBS-komentosarjakieltä, joka on yhteensopiva VBA-ohjelmointikielen kanssa. Lisäksi toteutuksessa käytettiin avuksi Excelin kehitystyökaluissa olevaa nauhuria. Sen avulla saatiin Excelin kentät ja komennot selville sekä tuloksen kirjoitus Exceliin toteutettua.

Automatisoidun simuloinnin tehtävänä on syöttää SAPin ZCU50-transaktiossa oleviin kenttiin kaikki vaadittava tieto simulointia varten. Lisäksi se kirjoittaa simulointituloksen Exceliin. Tässä tuleekin esiin makrokoodin suurin hyöty. Simulointi on mahdollista tehdä vain 1 moottori kerrallaan. Tämä on erittäin hidas tapa saada rakenteet ulos SAPista, mutta se tapahtuu silti automaattisesti. Käyttöliittymästä löytyykin kehotus jossa sanotaan, että isommat ajot kannattaa jättää yöksi tai viikonlopuksi pyörimään. Simulointia ei voi suorittaa taustalla, joten kenttien täyttäminen, simulointi ja kansioiden avaaminen näkyy koko ajan käyttäjän ruudulla. Kentät tosin täyttyvät niin nopeasti, että sen näkee vain välähdykseltä.

8.3.1 Käynnistäminen

Java-koodista siirtyminen VBA-koodiin on omanlaisensa prosessi. VBA-koodi on xlsm-päätteinen Excel-tiedosto. Se on siis rakennettu osaksi Excel-tiedostoa, mutta pelkkä Excel-tiedoston avaaminen ei käynnistä makrokoodia. Tästä syystä sen käynnistystä ei voitu tehdä suoranaisesti Javan puolelta. Java käynnistää sen sijaan pienen Visual Basic -koodinpätkän, joka osaa käynnistää makron. Ohjelmaa ajettaessa Excel-tiedosto ei avaudu näytölle, mutta tehtävienhallinnasta voidaan huomata EXCEL.EXE prosessin olevan päällä ajon aikana.

8.3.2 Tiedoston pilkkominen muuttujiin

VBA:n käynnistymisen jälkeen ensimmäinen tehtävä on lukea tekstitiedosto, joka luotiin Teamcenterin tietokannasta saatujen moottorikoodien, erottajan ja IM-koodin yhdistelmällä. Tekstitiedostoa lähdetään purkamaan lukemalla sitä rivi kerrallaan ja se on toteutettu for-silmukalla. Rivin lukemisen jälkeen koodi ajetaan läpi eli luodaan rakenne moottorille ja kirjoitetaan rakennerivit ylös Exceliin. Tämän jälkeen vasta siirrytään toiselle riville ja sama jatkuu niin kauan, kunnes tekstitiedosto on käyty kokonaan läpi.

Ensimmäinen pilkkominen tapahtuu siten, että koodi etsii rivistä kohdan ”IM:”, jonka tarkoitus oli alunperinkin vain jakaa teksti. Jakajamuuttujan alkioon nolla tallentuu erottajan vasen puoli eli tässä tapauksessa moottorikoodi. Tämä tallennetaan moottorikoodin merkkijonomuuttujaksi. Jakajamuuttujan ensimmäiseen alkioon tallentuu rivin oikea puoli, joka sisältää IM-koodin ja joka sekin sijoitetaan omaan merkkijonomuuttujaansa.

Toinen pilkkominen tapahtuu moottorikoodin merkkijonomuuttujassa, jos sen ehto toteutuu. Ehtona sillä on merkkijonon pituus. Pituuden ollessa yli 14 merkkiä suoritetaan pilkkominen siten, että katkaistaan koodi 14:n merkin kohdalta. Syy tähän on se, että moottorikoodi on tässä tapauksessa varianttikoodillinen moottorikoodi. Alkio nolla on nyt 14 merkkinen ja vain se jää moottorikoodin merkkijonomuuttujaan.

VBA ei tue kuitenkaan toimintoa, joka osaisi katkaista tekstin tietyn pituuden kohdalta. Tätä pilkontaa varten on luotu oma aliohjelma. Se toimii siten, että sille voi syöttää atribuutiksi kokonaisluvun, jonka se ottaa vastaan. Sen perusteella se osaa katkaista tekstin oikeasta kohdasta. Ylemmässä kohdassa sille annetiin atribuutiksi numero 14.

Varianttikoodit ovat 3 merkin sarjoissa. Yleensä niitä on vain 1, mutta jos moottorikoodissa on niitä enemmän, suoritetaan kolmas pilkkominen. Se tapahtuu siten, että äskeisen tekstin katkaisemisen oikea puoli, eli varianttikoodit, katkaistaan nyt 3 merkin välein. Se tapahtuu käyttämällä samaa aliohjelmaa kuin edellisissä koh-

dassa, mutta syöttämällä sille atribuutiksi numero 3. Aliohjelman tulos tallentuu varianttikoodien listamuuttujaan siten, että listan alkioihin tulee aina yksi 3 numeron sarja.

8.3.3 Simulointi

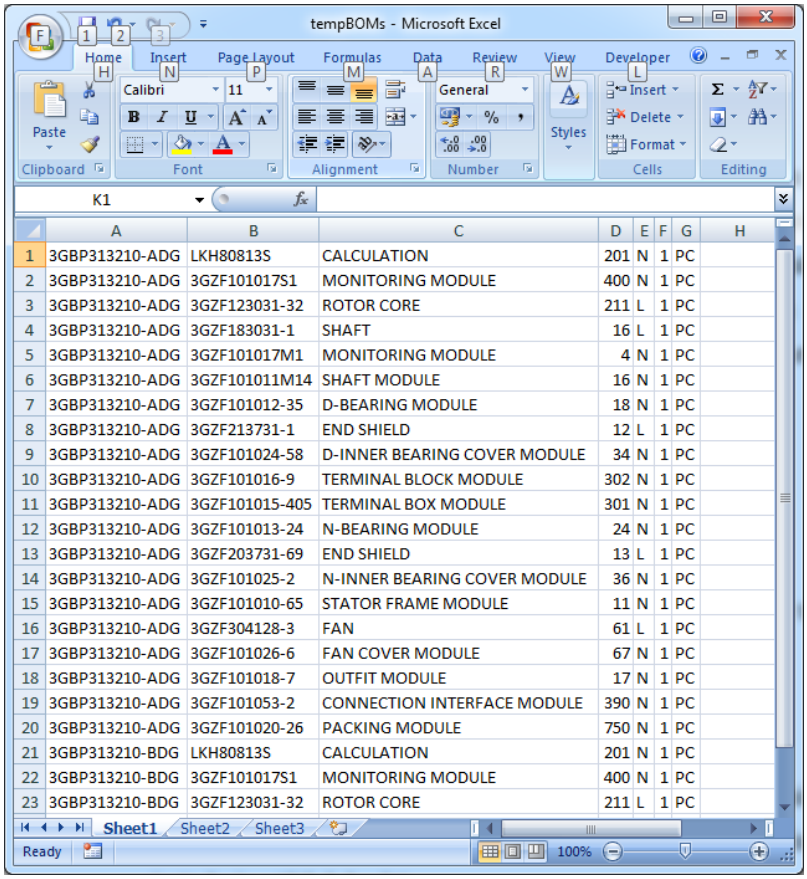
Ensimmäiseksi makrokoodi käynnistää ZCU50-transaktion. Seuraavaksi se täyttää transaktion kaikki kentät, koska jokainen kenttä on pakollinen simuloinnin kannalta. Kaikki kenttiin täytettävät tiedot ovat pilkkomisen ansiosta tallennettuna muuttujiksi. Plant-kenttään täytetään tosin aina numero 0800 ja Material-kohtaan syötetään aina 3G. Basic Code -kenttään tulee siis 14 merkkinen moottorikoodi ja IM Code -kenttään tietysti IM koodi. Variant Code -kentässä koodi avaa uuden ikkunan. Ikkunassa on kenttiä, joihin voidaan syöttää 3 merkin sarjoja. Varianttikoodit oli tallennettuna listamuuttujassa, joten se puretaan tässä kohtaa varianttikenttiin.

Kenttien täytön jälkeen makrokoodi suorittaa simulointi napin painamisen. Jos jossain kentässä on tässä kohtaa virhe tai transaktio ei voi suorittaa simulointia tälle moottorille, koodi hyppää virheenkäsittelijälle. Tämä kirjoittaa Exceliin tekstin ”Unable to simulate” ja tämän jälkeen siirrytään suoraan seuraavan moottorikoodin käsittelyyn.

8.3.4 Kansioiden avaaminen ja tuloksen tallentaminen

Simuloinnin valmistuttua näytölle avautuu moottorirakenteet, jotka ovat kansioiden sisällä. Nämä kansiot avataan myös makroa hyödyntämällä. Sen jälkeen voidaan tulos lukea kansioiden sisällä olevista nimikkeistä ja niiden rakennetiedoista.

Kuvasta 23 nähdään, kuinka moottorikoodit tallentuvat Excel-tiedostoon. Rivit rakentuvat sinne juuri halutulla tavalla, jotta sen lukeminen Javalla hajautuskarttaan olisi mahdollisimman helppoa. Java ei tue xlsm-päätteisten makrotiedostojen lukemista, joten lopputulos tallentuu lopuksi xlsx-päätteiseen tiedostoformaattiin.



	A	B	C	D	E	F	G	H
1	3GBP313210-ADG	LKH80813S	CALCULATION	201	N	1	PC	
2	3GBP313210-ADG	3GZF101017S1	MONITORING MODULE	400	N	1	PC	
3	3GBP313210-ADG	3GZF123031-32	ROTOR CORE	211	L	1	PC	
4	3GBP313210-ADG	3GZF183031-1	SHAFT	16	L	1	PC	
5	3GBP313210-ADG	3GZF101017M1	MONITORING MODULE	4	N	1	PC	
6	3GBP313210-ADG	3GZF101011M14	SHAFT MODULE	16	N	1	PC	
7	3GBP313210-ADG	3GZF101012-35	D-BEARING MODULE	18	N	1	PC	
8	3GBP313210-ADG	3GZF213731-1	END SHIELD	12	L	1	PC	
9	3GBP313210-ADG	3GZF101024-58	D-INNER BEARING COVER MODULE	34	N	1	PC	
10	3GBP313210-ADG	3GZF101016-9	TERMINAL BLOCK MODULE	302	N	1	PC	
11	3GBP313210-ADG	3GZF101015-405	TERMINAL BOX MODULE	301	N	1	PC	
12	3GBP313210-ADG	3GZF101013-24	N-BEARING MODULE	24	N	1	PC	
13	3GBP313210-ADG	3GZF203731-69	END SHIELD	13	L	1	PC	
14	3GBP313210-ADG	3GZF101025-2	N-INNER BEARING COVER MODULE	36	N	1	PC	
15	3GBP313210-ADG	3GZF101010-65	STATOR FRAME MODULE	11	N	1	PC	
16	3GBP313210-ADG	3GZF304128-3	FAN	61	L	1	PC	
17	3GBP313210-ADG	3GZF101026-6	FAN COVER MODULE	67	N	1	PC	
18	3GBP313210-ADG	3GZF101018-7	OUTFIT MODULE	17	N	1	PC	
19	3GBP313210-ADG	3GZF101053-2	CONNECTION INTERFACE MODULE	390	N	1	PC	
20	3GBP313210-ADG	3GZF101020-26	PACKING MODULE	750	N	1	PC	
21	3GBP313210-BDG	LKH80813S	CALCULATION	201	N	1	PC	
22	3GBP313210-BDG	3GZF101017S1	MONITORING MODULE	400	N	1	PC	
23	3GBP313210-BDG	3GZF123031-32	ROTOR CORE	211	L	1	PC	

Kuva 23. VBA koodin luoma väliaikaistiedosto.

8.4 Rakennerivien hakeminen SAPista

Todellisuudessa SAPin rakennerivien hakeminen tapahtuu VBA koodin luomasta Excel-tiedostosta. Tämän luokan ensimmäinen metodi käynnistää kuitenkin Visual Script -koodin, joka puolestaan käynnistää VBA-makrokoodin. Tämän jälkeen luokka jää odottelemaan, että makrokoodi on valmis, jonka jälkeen sen seuraavaa metodia kutsutaan.

Java API ei tue Excel formaattien lukemista. Se vaati Apache POIn asennuksen tietokoneelle, mutta vastaavasti olisi käynyt Java Excel APIn asennus. Asennuspaketista löydetyt jar-tiedostot mahdollistivat Excel-tiedostojen kirjoittamisen ja lukemisen. Tämän luokan tehtäväksi jää siis järjestelijän rooli ja lopuksi tuloksen poimiminen täsmälleen samanlaiseen muotoon kuin Teamcentrin puolella. Se tarkoitti hajautustaulua, johon luetaan avaimeksi moottorikoodi ja arvopariksi sen rakennerivit listana.

8.5 Rakennerivien vertailu ja lopputuloksen kirjoittaminen

Vertailu tapahtuu hajautustaulujen kesken, mitkä tämä luokka ottaa vastaan atribuutteina. Hajautustaulujen avaimet eli moottorikoodit ovat samat molemmissa järjestelmissä ja arvoparit eli moottorirakenteet ovat erilaiset. Tulostusta varten tarvitaan kaksiulotteinen merkkijonotaulukko, jossa ensimmäinen alkio vastaa Excelin rivinumeroa ja toinen alkio sarakkeen numeroa.

Vertailua lähdetään purkamaan toista hajautustaulua iteroimalla 1 moottorikoodi kerrallaan. Tämän jälkeen luodaan 2 listarakennetta, yksi SAPin ja yksi Teamcenterin rakenteita varten. Näihin listarakenteisiin haetaan sen hetkinen iteroitu moottorikoodi hajautustauluista. Tämä tarkoittaa sitä, että etsitään molemmista hajautustauluista sama moottorikoodi ja tallennetaan vastaavat rakennerivit listarakenteisiin. Jos esimerkiksi SAPin listaan tulee suurempi määrä rakennerivejä, lisätään Teamcenterin pienempään listaan sen verran rivejä, että listat ovat saman kokoiset.

Vertailu tapahtuu 3:ssa eri vaiheessa. Listarakenteet luetaan HashSet-kokoelmiin, joka mahdollistaa listojen vertailun. Ensin katsotaan mitkä rakennerivit ovat täsmälleen samat molemmissa listoissa ja kirjoitetaan ne Exceliin. Sen jälkeen kirjoitetaan SAP-listan rakenteet, jotka eivät täsmää Teamcenter-listan rakenteiden kanssa. Tässä kohtaa tekstiväri on värjätty punaiseksi. Sitten kirjoitetaan sama uudelleen, mutta rakenteet päin vastoin.

Hyvin useassa tapauksessa Teamcenterin hajautustaulussa löytyy moottorikoodille rakenne, mutta SAPin hajautustaulussa ei. Tämä on tapahtunut silloin, kun simulointi ei ole ollut mahdollista SAPissa. Tässä tapauksessa kirjoitetaan Exceliin molempien rakenteiden kohdalle teksti ”Unable to simulate” ja hypätään seuraavan iteroidun moottorikoodin käsittelyyn. Simulointikyvyttömät moottorikoodit merkitään Exceliin oranssilla väritunnisteella.

8.6 Muita luokkia

Muut luokat on luotu omiksi luokiksi lähinnä pienentääkseen pääluokan kokoa. Jokaisella niistä on kuitenkin tärkeä tehtävä ohjelman kannalta. Muut luokat ovat

kiertokyselynä toteutettu prosessien tarkkailu, väliaikaistiedostojen poistaja, käyttöliittymän valikko sekä ohjeet.

8.6.1 Tiedostojen poistaminen ja prosessien tarkkailu

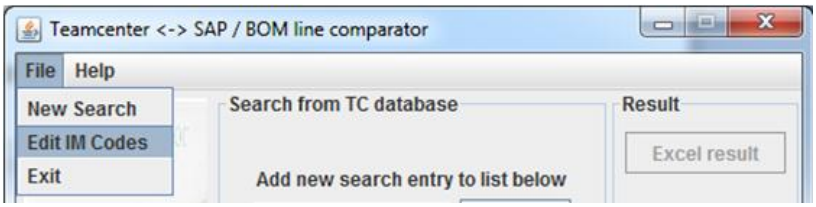
Väliaikaisten tiedostojen poistamiseen on oma luokkansa. Ohjelman ajon aikana se luo 2 väliaikaista tiedostoa, jotka ovat Teamcenterin haussa luotu tekstitiedosto, joka sisälsi moottorikoodit IM-koodineen, toinen oli Excel-tiedosto, joka saadaan ulos SAPista. Näitä ei olisi välttämätön edes poistaa, koska ohjelma osaa kirjoittaa samat tiedostot vanhojen päälle. Toisaalta, jos ohjelmakoodissa syntyy virhe ja se pystyy jatkamaan eteenpäin, voi se käyttää hyväksi vanhentunutta väliaikaistiedostoa. Kun sillä suoritetaan ohjelma loppuun, saadaan aivan varmasti väärä lopputulos. Tiedostojen poisto tapahtuu aivan viimeisenä vaiheena ohjelmakoodissa.

Luokka, jonka tehtävänä on tarkkailla prosesseja, koostuu kahdesta eri metodista. Ensimmäisen metodin tehtävä on SAPin tarkkailu. Ennen makrokoodin käynnistämistä se tekee prosessien tarkistuksen ja etsii sieltä SAPin prosessia. Jos SAP on päällä, ohjelma jatkaa normaalisti eteenpäin. Jos SAP ei ole päällä, ohjelma avaa viesti-ikkunan, jossa kehotetaan käynnistämään SAP Tuotanto ESP. Kun viestin OK-painiketta painetaan, tarkastaa se uudelleen prosessin ja niin kauan kuin SAP ei ole päällä, tulostuu viestikenttä uudelleen.

Toisena metodina se tarkkailee Exceliä. Makrokoodi on VBA-kieltä, joka on suoraan liitetty Exceliin. Excelin prosessia tarkkailemalla saadaan tietää, onko makro tehnyt työnsä loppuun. Prosessin kuuntelu käynnistetään vasta sen jälkeen, kun makro on jo työskentelemässä. Kuuntelu on toteutettu kiertokyselynä eli pollaamisena. Reilun puolen minuutin välein ohjelmakoodi tarkistaa Excel prosessin uudestaan ja tätä jatkuu niin kauan, kunnes prosessi huomataan suljetuksi. Makrokoodin päätyttyä, se sulkee automaattisesti kaikki avatut Excel-tiedostot, vaikka ne ovat ohjelman ulkopuolisia. Pollauksen huomatessa prosessin sammuneen, pääsee ohjelma jatkamaan eteenpäin.

8.6.2 Valikkorivi ja ohjeet

Ohjelman yläpalkissa löytyy valikkorivi ja se haetaan pääohjelman käyttöön omasta luokasta. Valikkoja on 2, joilla kummallakin on alavalikot. File-valikosta löytyy uuden etsinnän aloitus sekä siitä voidaan sulkea ohjelma. Kuvassa 24 voidaan nähdä valikon tärkein ominaisuus, joka on IM-koodien muokkaaminen. Sitä painettaessa ohjelma aukaisee ruudulle viesti-ikkunan, jossa on ohjeet IM-koditiedoston täytöstä. Kuittaamalla viesti OK-painikkeella, aukeaa varsinainen tekstitiedosto, jossa on kaikki klassifioinnista saadut IM-koodit. Tätä listaa voidaan joutua tulevaisuudessa täydentämään, kun klassifiointiin lisätään jotain.



Kuva 24. Valikkorivi.

Help-valikosta löytyvät ohjeet sekä englanniksi että suomeksi. Ohjeet sisältävät ohjelman käyttöohjeet ja varoitukset. Ohjeiden luokka on kirjoitettu myös omaksi luokaksi, ettei se vaikeuttaisi muiden luokkien tulkintaa. Tämä luokka sisältää käytännössä vain tekstiä.

9 SOVELLUKSEN TESTAUS

Ohjelmasta tehtiin jar-tiedosto, jolla se voidaan käynnistää ilman Eclipsen käyttöympäristöä. Kaikki Javalla tehdyt luokat pakkautuivat yhteen jar-tiedostoon. Kirjastot, jotka eivät ole osa Java APIa, kopioituivat omaksi kansioksi samaan hakemistoon kuin ohjelman käynnistystiedosto (Kuva 25.). Lisäksi tuli käsin kopioida makron käynnistys ja makron ajamiseen tarvittavat tiedostot sekä IM-koodien haku, kuva ja ohjeistus tiedostot. Nämä kaikki täytyy olla samassa kansiossa ohjelman toiminnan kannalta.

Kuva 25. Ohjelman tiedostot.

9.1 Alfatestaus ja virheiden korjaaminen

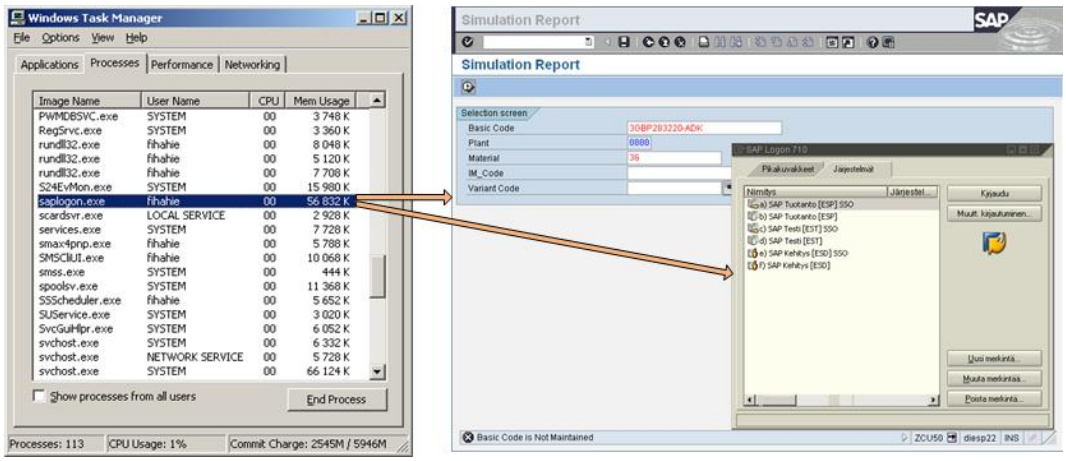
Jokaiseen ohjelman osa-alueeseen tehtiin ohjelmaa kirjoitettaessa väliaikaistestejä. Osa-alueita korjattiin ennen kuin seuraavaan kohtaan siirryttiin. Kun ohjelma oli saatu kokonaisuudessa pakettiin ja eri osa-alueet yhdistettiin, aloitettiin varsinainen alfatestaus. Se toteutettiin 3:lla eri metodilla. Yksi testaustapa oli hakea yksittäistä moottorikoodia Teamcenterin tietokannasta ja simuloida se SAPissa. Toinen tapa oli samantyylinen, mutta moottorikoodi oli sellainen, jota ei SAPissa voitu simuloida. Kolmas tapa oli hakea muutaman moottorin joukko Teamcenteristä, joka sisälsi simuloitavia sekä simuloimiseen kykenemättömiä moottorikoodeja. Näinkin pienen määrän simuloinnissa kesti useita minuutteja, joten ongelmakohtien ratkominen todellakin vaati kärsivällisyyttä.

9.1.1 Korjaamattomat virheet

Makrokoodin työskentely SAPissa aiheutti virheen, jos ennen simuloinnin alkamista kenttien täyttövaiheessa tai simuloinnin jälkeisessä kansion avaamisessa klikkaili hiirellä SAPin ikkunaa. Tämä johtui siitä, että makrokoodi toimii kuten käyttäjä. Jos hiiri on klikattu pois täyttöruudusta, kenttää ei voi tietenkään täyttää. Virheenkorjaaja toimii kenttien täytössä niin, että se hakee sen jälkeen seuraavan

moottorikoodin ja jatkaa siitä. Kansioiden avaamisvaiheessa koodi kuitenkin antaa virheen ja keskeytyy lopullisesti. Hyvää korjaustapaa tähän ei löydetty ja siitä tehtiin ohjelman ohjeisiin kielto, että SAPin käyttö on kielletty simuloinnin aikana.

Ohjelma tarkistaa ennen simulointivaihetta SAPin prosessin, jotta SAP olisi päällä ennen makron käynnistämistä. Kuva 26 paljastaa ongelman, joka on SAPin kirjautumisikkunassa. Se on SAPin varsinaisen käyttöliittymän kanssa samaa prosessia, joka huomataan tehtävienhallinasta. Pakottamalla tehtävienhallinnassa SAPin lopetus, lopettaa se siis molemmat ikkunat. Jos käyttäjällä on ainoastaan kirjautumisikkuna päällä, ohjelma yrittää jatkaa simulointivaiheeseen ja keskeytyy. Tässä ainoaksi vaihtoehdoksi muodostui käyttäjän informoiminen SAPin kirjautumisesta jo ennen varsinaista ohjelman käynnistystä. SAP kirjaa käyttäjän ulos itsestään, jos mitään tapahtumaa ei SAPissa ole tapahtunut hetkeen. Tämän vuoksi pelkkä kirjautumisikkuna on valitettavan usein käyttäjällä päällä.



Kuva 26. Kirjautumisikkuna ja käyttöliittymä ovat samaa prosessia.

Teamcenterin puolella, moottorikoodien haussa, voitiin havaita ylimääräisiä tuotekoodeja. Ne voivat olla testikoodeja tai asiakaskohtaisia moottorikoodeja. Nämä eivät noudata 3 merkin varianttikoodisääntöä, joten simulointivaiheessa varianttikentät täyttyvät väärin. Hakua on kuitenkin vaikea toteuttaa niin, että vain puhtaat moottorikoodit pääsisivät läpi. VBA-koodissa voisi tehdä ehtoja erikoiskoodeja koskien, mutta se jäi tältä erää tekemättä.

Suurimmat korjaamattomat virheet ovat kuitenkin ohjelman keskeytymisessä. Virheidenkäsittelijöitä ei koodissa hirveästi ole. Jos koodi jostain syystä keskeytyy, käyttäjälle ei tule siitä minkäänlaista ilmoitusta. Käyttäjä voi kyllä huomata keskeytyksen status-kentässä olevasta tekstistä, joka tulostuu väärin. Tämä tapahtuu ainakin silloin, kun tietokannasta etsitään yhtä moottoria, joka ei löydy sieltä.

9.1.2 Korjatut virheet

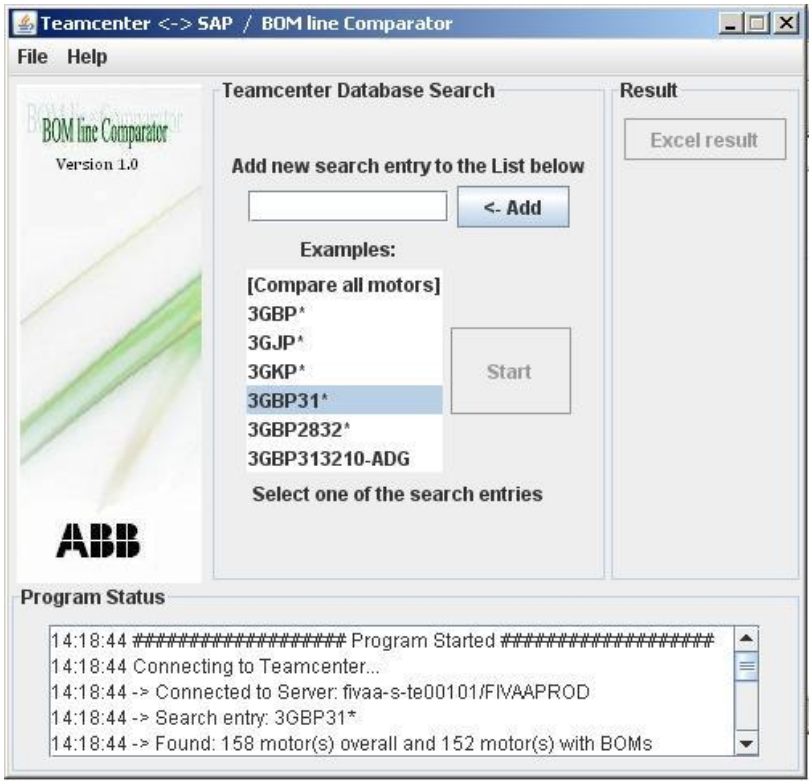
Korjattuja virheitä ei tähän testivaiheeseen mahtunut enää montaa, koska virheitä korjattiin osakokonaisuuksien päätteeksi. Lopullisessa tulostuksessa oli kuitenkin virhe. Se tulosti rakenteet täysin oikein, jos hakuna oli yksi moottorikoodi. Useamman moottoreiden haussa tulostus ei täsmännyt enää oikeita rakenteita. Tätä yritettiin korjata mitä erilaisin tavoin kokeilemalla eri ohjelmointimenetelmiä. Loppujen lopuksi virhe löytyikin Teamcenterin SQL-haku-luokasta, jossa rakennerivien tallennus hajautustauluun ei toiminut oikein. Sen perimmäinen syy oli rakennerivien nollauksen unohdus, kun kyselyn tulosta purettiin.

Juuri ennen simuloinnin aloitusta joudutaan ottamaan VBA:ssa varoitukset pois päältä, joten virheen sattuessa VBA ei anna minkäänlaista informaatiota. Tämä oli välttämätöntä siksi, että makrokoodi heittää ruudulle herjan silloin, kun se ei pääse etenemään tietyn aikavälin aikana. Tämä aikaraja ylittyy, kun koodi ei etene melkein kahteen minuuttiin transaktion hoitaessa simulointia. Varoitukset asetetaan takaisin päälle heti simuloinnin jälkeen, ennen kuin koodi tekee mitään muuta.

Status-kentässä oleva ajan näyttö ei toiminut, vaan ohjelma tulosti saman ajan jokaiseen kohtaan. Tämä saatiin korjattua siten, että tehtiin ajan lukemisesta oma metodi. Joka kerta, kun status-kenttään syötetään tekstiä, haetaan uusi aika metodista.

9.2 Massatestaus

Alkutestauksessa korjattujen virheiden jälkeen voidaan aloittaa massatestaus. Massatestauksessa on tavoitteena hakea Teamcenterin tietokannasta kaikki 3GBP31-alkuiset moottorikoodit ja simuloida niihin uudet rakenteet. Rakenteellisia moottorikoodeja löytyi tässä testissä 152 kappaletta (Kuva 27.).



Kuva 27. Massatestauksen aloitus.

Ohjelma jätetään rullaamaan yön ajaksi, koska simulointi vie paljon aikaa. Testin tavoitteena pyritään selvittämään kestääkö sovellus näin suuren määrän simuloitavia moottoreita ilman keskeytyksiä. Simulointiaika olisi hyvä myös saada selville. Tärkein tavoite testillä on kuitenkin lopputuloksen oikeellisuus, joka tarkistetaan pistokoemenetelmällä.

10 TULOKET

Kuvassa 28 on pätkä massatestauksen tuloksesta. Se näyttää päällisin puolin hyvältä ja tässä vaiheessa voitiin todeta, että sovellus ei keskeytynyt missään vaiheessa ja lopputulos saatiin kirjoitettua. Loppuaika näkyi ohjelman status-kentässä ja pienellä laskutoimituksella päästiin tulokseen alle kaksi tuntia. Myöskin rakenteet täsmäsivät niiden moottoreiden osalta molemmissa järjestelmissä, mitkä tarkistin. Tuloksessa löytyi myös moottorikoodeja, jotka eivät noudattaneet varianttisääntöä, mutta simuloitui silti rakenteeksi Exceliin.

	A	B	C	D	E	F	G	H	I
1	Teamcenter					SAP			
2	Motorcode	Item	Seq N	Q		Motorcode	Item	Seq N	Q
3	3GBP312220-ADG	3GZF101011M14	16	1		3GBP312220-ADG	3GZF101011M14	16	1
4	3GBP312220-ADG	3GZF101010-9	11	1		3GBP312220-ADG	3GZF101010-9	11	1
5	3GBP312220-ADG	3GZF101012-35	18	1		3GBP312220-ADG	3GZF101012-35	18	1
6	3GBP312220-ADG	3GZF101013-24	24	1		3GBP312220-ADG	3GZF101013-24	24	1
7	3GBP312220-ADG	3GZF101017M1	4	1		3GBP312220-ADG	3GZF101017M1	4	1
8	3GBP312220-ADG	3GZF101018-7	17	1		3GBP312220-ADG	3GZF101018-7	17	1
9	3GBP312220-ADG	3GZF101015-405	301	1		3GBP312220-ADG	3GZF101015-405	301	1
10	3GBP312220-ADG	3GZF101025-2	36	1		3GBP312220-ADG	3GZF101025-2	36	1
11	3GBP312220-ADG	3GZF101024-58	34	1		3GBP312220-ADG	3GZF101024-58	34	1
12	3GBP312220-ADG	3GZF101026-6	67	1		3GBP312220-ADG	3GZF101026-6	67	1
13	3GBP312220-ADG	3GZF101053-2	390	1		3GBP312220-ADG	3GZF101053-2	390	1
14	3GBP312220-ADG	3GZF213731-1	12	1		3GBP312220-ADG	3GZF213731-1	12	1
15	3GBP312220-ADG	3GZF203731-69	13	1		3GBP312220-ADG	3GZF203731-69	13	1
16	3GBP312220-ADG	3GZF304128-3	61	1		3GBP312220-ADG	3GZF304128-3	61	1
17	3GBP312220-ADG	3GZF101016-615	302	1		3GBP312220-ADG	3GZF101016-9	302	1
18	3GBP312220-ADG	3GZF121031-32	15	1		3GBP312220-ADG	3GZF101017S1	400	1
19	3GBP312220-ADG	3GZF131031-35	200	1		3GBP312220-ADG	3GZF101020-26	750	1
20	3GBP312220-ADG	3GZF374731-19	750	1		3GBP312220-ADG	3GZF123031-26	211	1
21	3GBP312220-ADG	EmptyBomLine1	999	999		3GBP312220-ADG	3GZF183031-1	16	1
22	3GBP312220-ADG	EmptyBomLine2	999	999		3GBP312220-ADG	LKH80641S	201	1
23	Motorcode	Item	Seq No.	Qty		Motorcode	Item	Seq No.	Qty
24	3GBP312240-ADK	Unable to Simulate	-------	---		3GBP312240-ADK	Unable to Simulate	-------	---
25	Motorcode	Item	Seq No.	Qty		Motorcode	Item	Seq No.	Qty
26	3GBP313230-ADG	3GZF101010-9	11	1		3GBP313230-ADG	3GZF101010-9	11	1
27	3GBP313230-ADG	3GZF101011M14	16	1		3GBP313230-ADG	3GZF101011M14	16	1
28	3GBP313230-ADG	3GZF101012-35	18	1		3GBP313230-ADG	3GZF101012-35	18	1
29	3GBP313230-ADG	3GZF101013-24	24	1		3GBP313230-ADG	3GZF101013-24	24	1
30	3GBP313230-ADG	3GZF101018-7	17	1		3GBP313230-ADG	3GZF101018-7	17	1
31	3GBP313230-ADG	3GZF101017M1	4	1		3GBP313230-ADG	3GZF101017M1	4	1
32	3GBP313230-ADG	3GZF101015-405	301	1		3GBP313230-ADG	3GZF101015-405	301	1
33	3GBP313230-ADG	3GZF101025-2	36	1		3GBP313230-ADG	3GZF101025-2	36	1
34	3GBP313230-ADG	3GZF101024-58	34	1		3GBP313230-ADG	3GZF101024-58	34	1
35	3GBP313230-ADG	3GZF101026-6	67	1		3GBP313230-ADG	3GZF101026-6	67	1
36	3GBP313230-ADG	3GZF101053-2	390	1		3GBP313230-ADG	3GZF101053-2	390	1
37	3GBP313230-ADG	3GZF203731-69	13	1		3GBP313230-ADG	3GZF203731-69	13	1
38	3GBP313230-ADG	3GZF213731-1	12	1		3GBP313230-ADG	3GZF213731-1	12	1
39	3GBP313230-ADG	3GZF304128-3	61	1		3GBP313230-ADG	3GZF304128-3	61	1
40	3GBP313230-ADG	3GZF101016-615	302	1		3GBP313230-ADG	3GZF101016-9	302	1

Kuva 28. Massatestauksen tulos.

Miinuksena koin hieman sen, että simuloituja moottoreita oli reilu 40 kappaletta ja simulointikyvyttömiä reilu 100. Tähän ei tietenkään voi itse vaikuttaa, vaan se johtuu ZCU50-transaktion rajoittuneisuudesta tai IM-koodien saatavuudesta klassifioinnista. Tästä syystä simulointi kesti vain hetken ja ohjelmaa ei olisi tarvinnut yöksi jättää rullaamaan. Isomman moottorimäärän testausmenetelmä olisi tuonut lisää varmistusta ohjelman luotettavuudesta.

Tuloksessa samanlaiset rakennerivit ovat kohdakkain, mutta virheelliset rakennerivit ovat sattumanvaraisessa järjestyksessä. Se vaikeuttaa hieman rakenteiden vertailua. Tämän voisi korjata, jos jokainen rakenne voitaisiin laittaa aakkosjärjestykseen eli sortata. Tätä ei ainakaan Javasta käsin voinut tehdä, mutta Excelissä se voisi ehkä olla mahdollista käsin tai makron avulla.

Kuvaa 28 tutkimalla nähdään, että ensimmäisessä moottorissa rakenneriveistä 6 riviä ei täsmää. Tässä huomataan heti, että Teamcenterin kannasta tulee 2 rakenneriviä vähemmän. Tämä johtuu siitä, että järjestelmissä osaluettelorakenne on luotu hieman eri tavalla. Eriävissä rakenneriveissä ei myöskään täsmää jokainen osanumero keskenään, mikä johtuu myös samasta syystä. Osanumero 750 on molemmissa rakenteissa, mutta Teamcenterin puolella se on komponentin nimikkeellä ja SAPissa moduulina. Osanumero 302 paljastaa todellisen rakennevirheen.

11 YHTEENVETO

Työ tehtiin ABB Oy Motors and Generatorsin järjestelmäkehitysosastolle, jonka tavoitteena on edistää Teamcenterin toimivuutta. Työssä valmistunut työkalu on tarkoitettu kuitenkin pääosin suunnittelijoiden käyttöön. Työkalu vertailee tuotannonohjausjärjestelmässä luotuja moottorirakenteita suunnittelu- ja tuotetiedon hallintajärjestelmässä oleviin moottorirakenteisiin, jotka uusien sääntöjen johdosta saattaavat sisältää virheitä.

Projekti aloitettiin tutkimalla työn ideaa ja miettimällä mahdollisia toteutustapoja ongelmien ratkaisemiseksi. Esitutkintavaiheeseen voitaisiin lukea myös järjestelmiin paneutuminen, jonka tarkoituksena oli selvittää eritoten vertailukohteiden saaminen tietokannoista. Näiden kohdalle voisikin vetää pisimmän aikajanan koko projektin elinkaaressa.

Esitutkinnan jälkeen projekti voitaisiin jakaa 4 suurempaan osa-alueeseen, joita rajaa eri järjestelmät ja ohjelmointikielien käytöt. Projekti alkoi SQL-kyselyillä erillisellä ohjelmalla ja tarkoituksena oli löytää moottorit ja niiden rakenteet Teamcenter-järjestelmästä. Ongelman ratkettua toteutettiin sama kysely ohjelmakoodin sisällä. Tässä vaikeuksia tuotti yhdistäminen tietokantaan. Sen vaikeudet oli lisäkirjastojen vaiheet lataamisesta käytäntöön sekä selvitys palvelimen osoitteesta, käyttäjänimestä ja salasanasta. Tästä kuitenkin selvittiin yhteistyön turvin.

Toinen, ja ehkä ongelmallisin vaihe, oli SAP-järjestelmässä olevan ZCU50-transaktion automatisoiminen. Sen tarkoituksena oli luoda uudet moottorirakenteet Teamcenter-järjestelmästä saatujen moottorikoodien perusteella. Ongelmia tuotti se, että VBA:lla ohjelmoiminen ei ollut ennestään tuttua. Syntaksiin ja ohjelmointiparadigmaan tutustuminen otti oman aikansa. Tämä osittain vaikeutti ja hidasti projektin etenemistä, mutta uuden oppiminen on kaiketi rikkaus.

Järjestelmien ongelmat olivat hyvin erilaisia, vaikka niiden päämäärä oli yhtenäinen. Erilaiset järjestelmärakenteet, toteutustavat ja ohjelmointikielet tekivät rakennerivien hakemisesta omanlaisensa projektit. Hakutulos oli kuitenkin molemmissa järjestelmissä sama, moottorirakeneet saatiin niistä ulos. Niiden mittasuh-

teet olivat silti varsin erilaiset. Jos Teamcenterin puolella moottorirakenteiden hakemiseen menee 5 sekuntia, vie tämä SAPin puolella kutakuinkin 5 tuntia niiden luomiseen.

Kolmannessa vaiheessa samaan pakettiin yhdistetään rakenteiden vertailu ja tuloksen esittäminen Excel-taulukossa. Rakenteet sijoitetaan hajautustauluihin, joita vertailemalla löydetään eriävät rakenteet järjestelmien välillä. Koko projektin suurin yksittäinen ongelma tulikin tähän kohtaan, kun projektin piti olla jo loppusuoralla. Tulostus ei toiminut oikealla tavalla. Ongelmaa hankaloitti se, että vertailun läpikäymisen ja tulostuksen seasta virheitä oli vaikeampi paikantaa. Virhe oli arvatenkin yksi vaivainen rivi, mutta sen korjaaminen kesti viikkoja.

Neljäs vaihe oli rakentaa yksinkertainen graafinen käyttöliittymä, jota ulkopuolinen henkilö osaisi käyttää. Tämän toteuttaminen tapahtui palanen kerrallaan, eikä tässäkään ongelmilta vältytty. Näitä olivat pienet kitkat ruudun sovittamisen ja erilaisten objektien lisäämisen välillä. Se sai päätöksen useiden eri väärin tehtyjen tapojen kautta. Neljäs vaihe käsitti myös pääohjelman, jonka tehtävänä oli suorittaa itse ohjelma ja kontrolloida sen eri osia. Koodit on jaettu moneen eri palaseen, mutta kaikki sen palaset yhdistyvät pääohjelmassa. Vaikeuksia tuotti eniten ohjelmakoodista toiseen vaihtaminen. Sitä ei voitu toteuttaa niin kuin oli suunniteltu ja toteutustavaksi jäi prosessin kuunteleminen kiertokyselymenetelmällä.

Projekti saatiin onnistuneesti päätökseen ja kaikki sen kriittiset osa-alueet myöskin toteutettua. Sen toteutus vaati monta hikipisaraa, mutta ongelmien ratkominen teki työstä myös mielenkiintoisen. Uuden ohjelman tuottaminen, uusien toteutustapojen ja loogisten ratkaisujen kehittäminen asetti omat haasteensa, mutta olivat samalla myös innostavaa ja opettavaista. Jos ohjelmaideaa viedään tulevaisuudessa pidemmälle, voidaan siitä saada monipuolinen työkalu eri käyttötarkoituksiin. Ensimmäinen ja toivottavasti ei viimeinen versio rakennerivien vertailijasta on saatu pakettiin.

12 JOHTOPÄÄTÖKSET

Ohjelman toteutustapaan oli annettu vapaat kädet, joten päämäärän tavoittelu sai tapahtua keinoja kaihtamatta. Tavoitteisiin päästiin, koska välttämättömät vaatimukset saavutettiin ja ohjelma tekee sen, mitä sen pitääkin tehdä. Laajennettavuutta heikentää toki eri ohjelmointikielien käyttäminen. Sovellusprosessin elinkaari on vielä kesken, koska sen käyttöönottoa ei virallisesti ole vielä toteutettu. Toteutuksen jälkeen ylläpidon merkitys korostuu.

Parannusehdotuksia on tullut ilmoille työn edetessä ja sen valmistuttua. Parin ylimääräisen mutkan välttämiseksi ohjelma voitaisiin toteuttaa esimerkiksi pelkästään VBA-ohjelmointikielellä. Se jäi omien rajallisten taitojen takia toteuttamatta.

SAPiin olisi voinut toteuttaa automaattikirjautumisen sen sijaan, että ohjelma tarkistaa vain sen käynnissäolon ja antaa ilmoituksen tästä. Automaattikirjautumiseen löytyi kyllä ohjeita Internetistä, mutta sitä ei saatu toteutettua ABB:n SAP-ympäristössä. Makrokoodin työskennellessä SAPissa, ohjelma odottaa tätä prosessia kiertokyselynä. Javassa olisi prosessin odotukselle oma funktio, joka olisi varmasti ollut järkevämpi ratkaisu tässä kohtaa. Ongelmana oli se, että odotus toimi vaan käynnistettävälle prosessille, joka oli tässä tapauksessa vain pieni Visual Basic -koodinpätkä.

Yksinkertainen parannus olisi ilmoittaa lopputuloksessa simuloitujen ja simuloitumattomien moottoreiden määrä. Myös eriävien rakennerivien määrän olisi voinut ilmoittaa jokaisen rakenteen kohdalla. Lisäksi olisi voinut käyttää Excelissä eri välilehtiä, joihin olisi voinut tulostaa rakenteet eri tavoin.

Lähitulevaisuuden jatkotoimiin kuuluu palautteen kuunteleminen. Sen, kuinka hyödyllinen ohjelma on nyt ja kuinka hyödyllisen siitä voi vielä tehdä, päättää viime kädessä sen käyttäjät. Ohjelmaa ei ole vielä julkaistu suunnittelijoiden käyttöön, joten lähitulevaisuus on kriittinen ohjelman kannalta. Tämä vaihe onkin ohjelman betatestausvaihe. Palautteen vastaanotto, uusien ideoiden luominen ryhmässä, uusien näkökantojen kuunteleminen ja uudelleen ohjelmoiminen voisi olla seuraavan version tavoite.

Siemensin valmistaman Teamcenter-järjestelmän mukana tuli heidän kehittämä latausohjelma, joka toimii tekstitiedostoa täyttämällä ja ajamalla 1 batch-tiedosto. Tämän latausohjelman avulla voidaan lähettää Teamcenterin tietokantaan lomakkeiden kentistä klassifiointiin asti haluttua dataa. Sen käyttäminen ei ole kuitenkaan niin suotavaa, koska se ylikirjoittaa vanhan datan. Siinä piilee siis omat riskinsä. Ajatuksiin on kuitenkin tullut idea, että tämän työn sovellusohjelma kirjoittaisi valmiin lataustekstitiedoston virheellisten moottorirakenteiden korjaamista varten. Samalla ohjelma voisi myös ajaa korjatut rakenteet sen tietokantaan, mutta suotavampaa olisi varmasti tarkistaa korjaukset ennen ajoa.

Yksi idea saatiin tuotekehitysosaston puolelta. Jatkokäyttöä ajatellen ohjelman voisi toteuttaa vertailemaan mitä tahansa yksittäistä osaa järjestelmien tietokantojen välillä. Tämä loisi sovellukselle isomman käyttäjäkunnan. Toinen mahdollinen toteutus olisi vertailukohteiden hakeminen Teamcenterin klassifioinnista.

Tulevaisuutta on mahdoton ennustaa, mutta uudet 64-bittiset Windows 7 -käyttöjärjestelmät ovat päivittymässä Motorsille. Ohjelmistoihin ajetaan uusia päivityksiä päivittäin. Teamcenter tullaan päivittämään tulevaisuudessa uuteen versioon. Tulevaisuus näyttää, toimiiko sovellus päivitetyssäkin järjestelmässä. Tällaisia tilanteita varten ohjelma suunniteltiin kuitenkin muokattavaksi ja sen osa-alueet on kommentoitu kenen tahansa käyttöön.

LÄHDELUETTELO

/1/ 2K mediat 2000-2012. Johdatus SQL:n maailmaan. Viitattu 20.11.2012. Saatavilla www-muodossa: http://www.2kmediat.com/sql/alkeet.asp

/2/ ABB Oy Motors 2006. General technical specification – Mounting arrangements. Viitattu 28.11.2012. Saatavilla www-muodossa: http://abblibrary.abb.com/global/scot/scot234.nsf/veritydisplay/0c725072381e6cbec125784f0037fbc9/$file/ppm_%205%20lv%20motors%20for%20high%20ambient%20temperatures%20%20en%2012-2006.pdf

/3/ ABB Oy 2011 Oppaat, ohjeet ja yleistietoa – Henkilöstöopas: Tervetuloa taloon. Viitattu 2.8.2012. Saatavilla www-muodossa: http://search.abb.com/library/Download.aspx?DocumentID=9AKK105713A2265&LanguageCode=fi&DocumentPartId=&Action=Launch

/4/ ABB Oy Motors and Generators 2011. Esitysmateriaalit – kalvosarja. Viitattu 13.9.2012. Saatavilla www-muodossa: http://fi.inside.abb.com/cawp/gad00195/d3c443f55a868a2dc2257005001a8c24.aspx

/5/ ABB Oy Motors and Generators 2012. Kalvosarjat – Esitys moottoreiden moduulirakenteesta. Ei julkisesti saatavilla.

/6/ About.com 2012. Visual Basic. Viitattu 10.11.2012. Saatavilla www-muodossa: http://visualbasic.about.com/od/vbnetspecialtopics/a/aboutvb.htm

/7/ Free Java Guide 2006-2009. History of Java programming language. Viitattu 20.11.2012. Saatavilla www-muodossa: http://www.freejavaguide.com/history.html

/8/ Java 2012. Learn About Java Technology. Viitattu 14.10.2012 Saatavilla www-muodossa: http://www.java.com/en/about//

/9/ Karjalainen, J. ABB Oy 2012. Teamcenter ohje, FI. Ei julkisesti saatavilla.

/10/ Laine, J. & Luoma, M. 2009. SAP perusteet, navigointi, kyselyt. ABB:n sisäinen ohje FIMOT 1590. Ei julkisesti saatavilla.

/11/ Microsoft Corporation 2012. Microsoft tukipalvelut. Viitattu 11.11.2012 Saatavilla www-muodossa: http://office.microsoft.com/fi-fi/word-help/makron-kirjoittaminen-tai-nauhoittaminen-HA010099769.aspx

/12/ Oracle 2005. New to Java Programming Center – Unraveling Java Terminology. Viitattu 13.10.2012. Saatavilla www-muodossa: http://www.oracle.com/technetwork/topics/newtojava/unravelingjava-142250.html

/13/ Oracle 2009. Oracle Press Release – Oracle Buys Sun. Viitattu 23.11.2012. Saatavilla www-muodossa: http://www.oracle.com/us/corporate/press/018363

/15/ The Eclipse Foundation 2012. About the Eclipse Foundation. Viitattu 7.11.2012. Saatavilla www-muodossa: http://www.eclipse.org/org/#

/16/ The Eclipse Foundation 2012. FAQ Where did Eclipse come from? Viitattu 7.11.2012. Saatavilla www-muodossa: http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F

LIITTEET

Liite 1: TCS Confidental – processing logic

TCS Confidental - Processing Logic

The input screen takes the basic code characteristics, material, plant (0800) and if the other characteristics required which are available on the selection screen.

The basic code is validated against the entry in Z-table ‘ZVC_T_BASICCODE’.

Material is validated against ‘3G’ or ‘3GA’.

The user authorization against the material is checked in Z-table ‘ZVC_T_USERAUTH’.

The corresponding sales order is also fetched for the material entered and current user.

Fetch the data using the below process for the Characteristics “MC_VARIANT_CODE” and “MC_IM_CODE” using the given input Characteristic values.

For the given input material and Characteristics, update the variant Configuration using the FM “CE_C_PROCESSING” for the BOM Application 'ZMOT'.

Gets the instance number using FM “CUKO_DDB_HAS_ROOT”. Explode the BOM

using FM “CAVC_C_EXECUTE”.

Set the characteristic values using FM “CAVC_I_CHARS_SET_VALUES” for the instance number generated.

Use BAPI “BAPI_CFGINST_CHARCS_VALS_READ” to get current values of a characteristic.

Get all the child Instances using FM “CAVC_I_SELECT_CHILDS”.

Configure the Engineering BOM using FM “CE_C_PROCESSING”.

Read the Data (BOM Component, sort string, item category, component quantity, component quantity) of an Instance (BOM Item) using the FM CAVC_I_GET_BOM_ITEM_DATA. If the Instance Number is not generated, display a message Error message “No Data found” and exit the program.

Then the data as fetched for display using the below FM’s ‘CAVC_O_ORDER_BOM_INIT’ to get the parent details, ‘CAVC_I_GET_BOM_ITEM_DATA’ to get the sort string details,

‘CAVC_I_SELECT_CHILDS’ to get the childs for the particular material.

With these FM the data is prepared in form of parent child relationship.

The childs are extracted till the variant are reached i.e. 3 levels for rotors and stator and one level for modules.

The classification data is populated using the FM ‘CLAF_CLASSIFICATION_OF_OBJECTS’ for the materials